REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
14-07-2009	Final Technical	01-11-2007 to 30-04-2009

4. TITLE AND SUBTITLE
(U)DEVELOPMENT OF PRIME CYBER-INFRASTRUCTURE FOR COMBUSTION

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-08-1-0003

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
Michael Frenklach and Michael Gutkin

5d. PROJECT NUMBER
2308

5e. TASK NUMBER
BX

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California at Berkeley
Department of Mechanical Engineering
Berkeley, CA 94720-1740

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
875 North Randolph Street
Suite 325, Room 3112
Arlington, VA 22203-1768

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The initiative named PrIMe (for Process Informatics Model) is designed to keep track of models, model parameters, and experimental data in a global, integrated framework for the field of Combustion. It is aimed at curation of community data with the objective of collaborative development of reaction mechanisms of scientific explorations and predictive models for practical systems. This project provided funding for additional human resources, a professional programmer, who extended the development of the PrIMe cyber-infrastructure by developing one of its central parts, PrIMe Workflow Application. The Report presents a detailed description of the software developed.

15. SUBJECT TERMS
Modeling, combustion, global systems, web-based application, collaborative science

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Julian Tishkoff
Unclassified	Unclassified	Unclassified	UL	49	**19b. TELEPHONE NUMBER** *(include area code)* (703) 696-8478

Development of PrIMe Cyber-Infrastructure for Combustion

Michael Frenklach and Michael Gutkin

Final Technical Report to AFOSR
July 14, 2009

Department of Mechanical Engineering
University of California
Berkeley, CA 94720

Table of contents

1 Introduction

PrIMe (Process Informatics Model) is a new approach for developing predictive models of chemical reaction systems that is based on the scientific collaboratory paradigm and takes full advantage of existing and developing cyber infrastructure. The primary goals of PrIMe are collecting and storing data, validating the data and quantifying uncertainties, and assembling the data into predictive models with quantified uncertainties to meet specific user requirements. The principal components of PrIMe include: a data Depository, which is a repository of data provided by the community, a data Library for storage of evaluated data, and a set of computer-based tools to process data and to assemble data into predictive models. Two guiding principles of PrIMe are: open membership—a qualified individual or industrial organization can register to participate in the project; and open source—all submitted data, tools and models will be in the public domain

1.1 Purpose of the document

The current document describes the PrIMe Workflow Application architecture and its internal structure. Component functionality is depicted in the form of Use Case diagrams. Class diagrams, consistency diagrams, data-base scheme and components diagrams are used to demonstrate system design and component interaction.

The document consists of the following chapters:

1. The common system structure and the purpose of its modules.
2. PrIMe portal description and functionality.
3. Component Uploader general architecture description and functionality.
4. PrIMe Workflow Application general architecture description and functionality.

2 The common system structure

The common structure of PrIMe is shown in Figure 1. It consists of the following components:

1. **PrIMe portal** is responsible for system user management. It implements user authorization and authentication services, assigns user roles, and manages user permissions. Additionally, it enables users to collaborate. In the PrIMe portal you can find information concerning the latest changes, documentation, operating instructions and so on. The site is a portal to two additional systems—Scientific Component Uploader and PrIMe Workflow Application.

2. **Scientific Component Uploader (SCU)** is used to develop and deploy new scientific components. It allows the scientific component developer to upload a new scientific component, assign resources to components, and edit properties and configuration information of his/her previously developed components. All changes made by developers are stored as separate revisions, allowing a developer to open and edit any existing revisions. Only a user with administrator privileges can create a new revision and deploy it to the PrIMe Workflow Application (PWA).

3. **PrIMe Workflow Application (PWA)** is the site where a user works with scientific workflow projects. The scientific workflow project is built using preconfigured scientific components which are linked together in a network. The user can set input and output information for each scientific component and if applicable, the user can set configurable

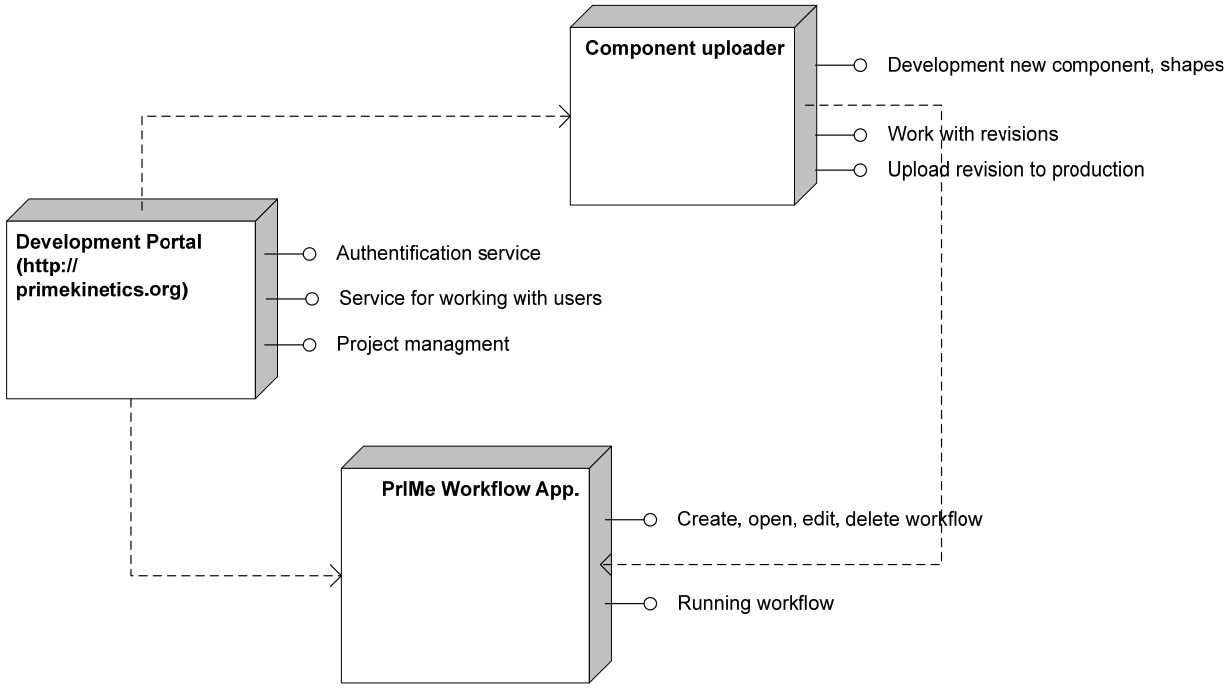

Figure 1. Components of the PrIMe structure

properties of a scientific component. In the PrIMe Workflow Application a user can create new scientific workflow projects, open existing projects, and execute valid workflow projects.

3 PrIMe portal

Purpose

The portal administers user management functions and stores workflow project information. The main system Use Case is represented in Figure 2.

System functions:

1. *User management functionality*. The PrIMe portal implements user authentication and authorization. Additionally, it assigns roles to each user and grants permissions to view/edit the users previously existing workflow projects and scientific components.

2. *Content management functionality*. The PrIMe portal manages all application documentation. This includes the scientific components manuals, system structure changes, and information concerning development of new scientific components.

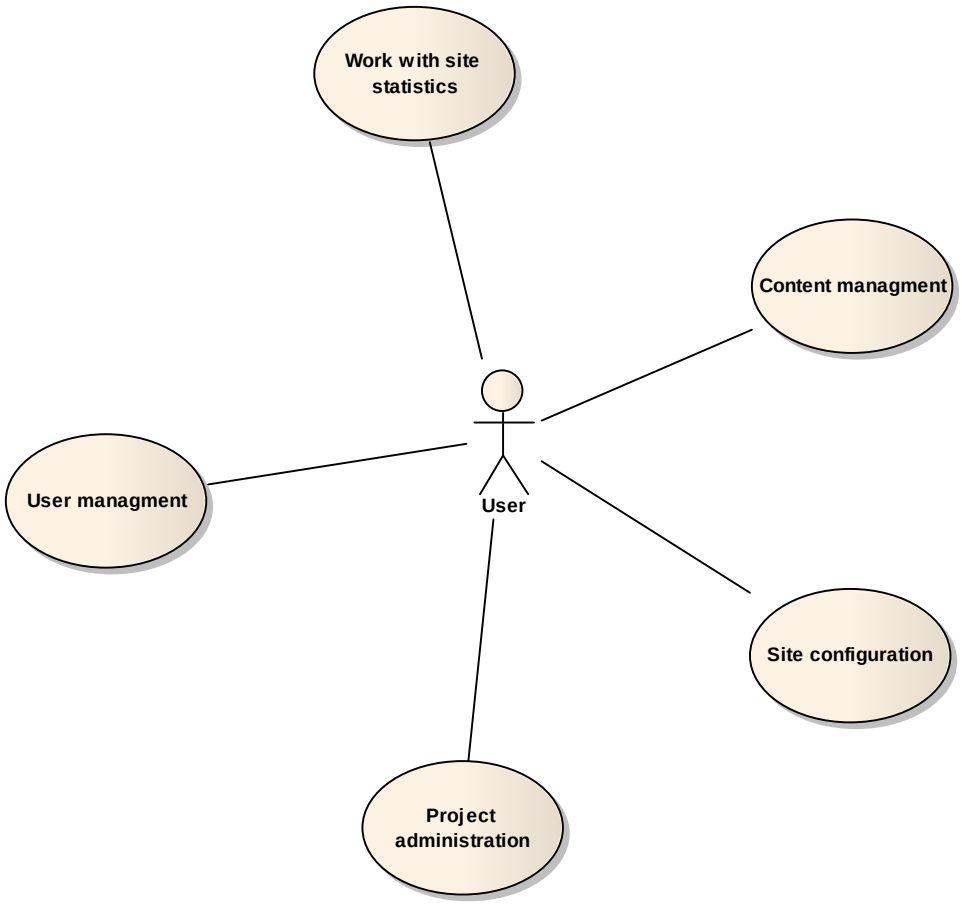

Figure 2. PrIMe portal Use Case diagram

3. *Authentication/authorization management for PWA and SCU*. The PrIMe portal authenticates, authorizes, and assigns user roles allowing a user to access PWA and SCU. This is done automatically as the user navigates to the PWA and SCU.

4 Scientific Component Uploader (SCU)

The main purpose of the Scientific Component Uploader is to facilitate the development and deployment of scientific components. It enables development of scientific components by managing existing component revisions, component properties, component resources, configuration information, and allows creation of new scientific components. The scientific component uploader also allows a developer to test the component before deployment to the PWA.

4.1 System Architecture

System Architecture is shown in Figure 3. The following main components are represented:

1. Client browser. The client browser allows the scientific component developer to configure, test, and deploy a scientific component. The SCU utilizes Microsoft Active X technology which allows executing the libraries native code in the browser context. Upon loading the SCU, the Manager.dll library is copied to the user's computer. This library provides all the necessary functionality for development, testing, and deployment of scientific components.
2. Server. The SCU is located on the server. When a user enters the SCU Manager.dll is copied to his client machine and is executed in the context of the client browser. The browser communicates with the server via web-services. All of the functions of the server are accessed through the Utility.dll library. The functions of the database, and user authorization and authentication are implemented in Utility.dll. The web-service is located on the server, which provides the interface for communication between the Manager.dll library, which is executed on the client, and the Utility.dll library which is executed on the server.
3. PrIMe Workflow Application. After the scientific component has undergone testing and is ready for deployment it is uploaded to the PrIMe Workflow Application to allow other PrIMe users to include the newly deployed scientific component in workflow projects.

SCU is connected with PWA via web-services. An administrator deploys a newly developed scientific component to the PWA by means of an appropriate web method.

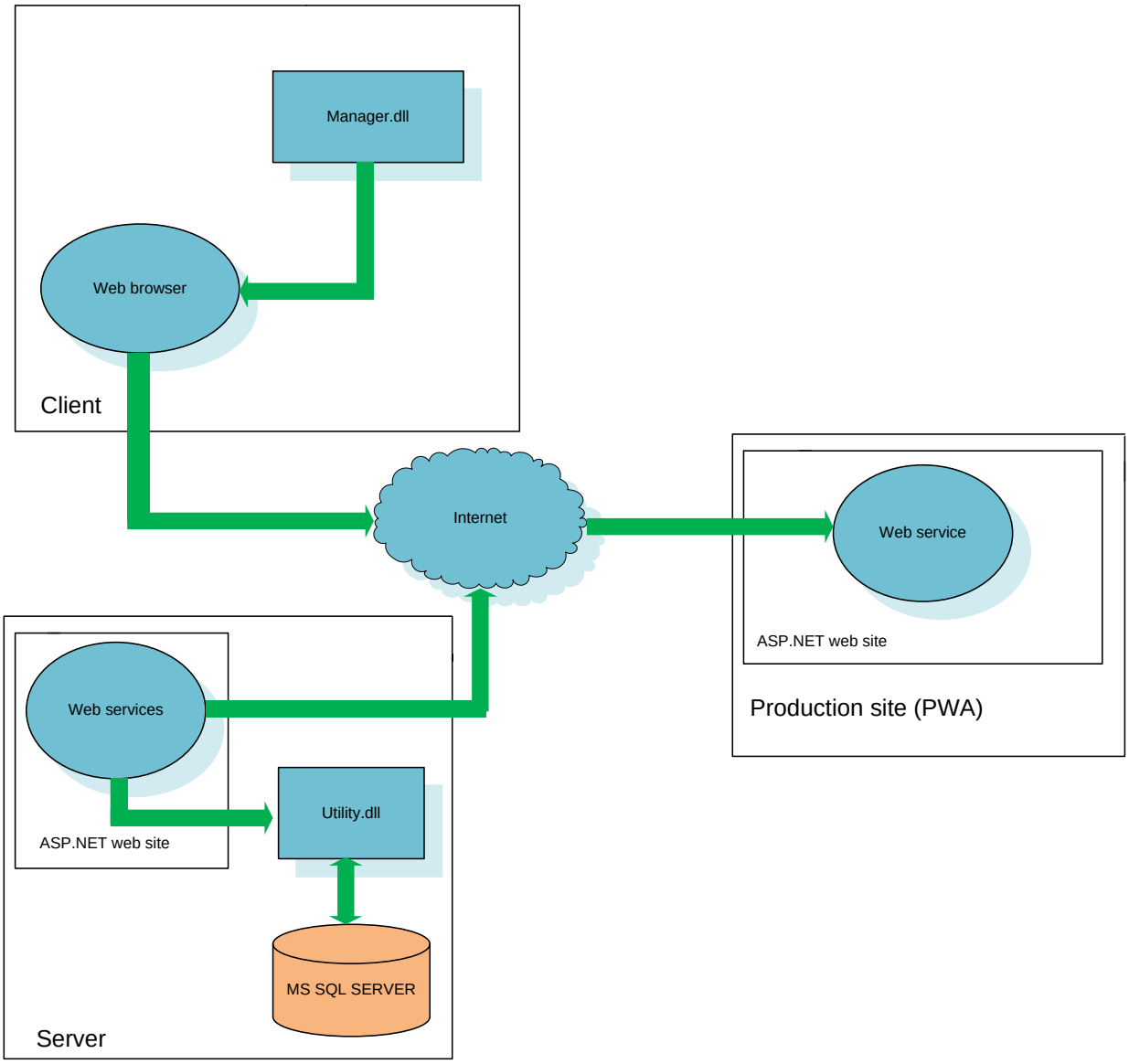

Figure 3. Scientific Component Uploader in the context of the client and PWA

4.2 Use Case systems

The SCU manages scientific components, resources, and component configuration information. The main functions of the SCU are listed below and the SCU Use Cases are represented in Figure 4.

Main functions:

1. *Manage component resources*. The user can upload new component images, edit images, and remove existing images associated with a scientific component from the server.
2. *Manage scientific components*. The SCU enables the user to add, remove, and edit scientific components.

3. *Configure scientific components*. The user can configure scientific components by adding, editing, and removing component inputs, outputs and other properties.
4. *Store scientific component location*. If the scientific component is a remote type, the SCU points to the server where the remote scientific component is executed from.
5. *Manage component revisions*. The SCU captures and saves all changes made by the user when editing components, resources, and configuration information as revisions. The MS SQL server stores all revision information.
6. *Scientific component testing*. The SCU allows the scientific component developer to test his/her component and confirm that it will work appropriately with PWA.
7. *Scientific component deployment*. The SCU allows an administrator to deploy any revision of a scientific component to the PWA.

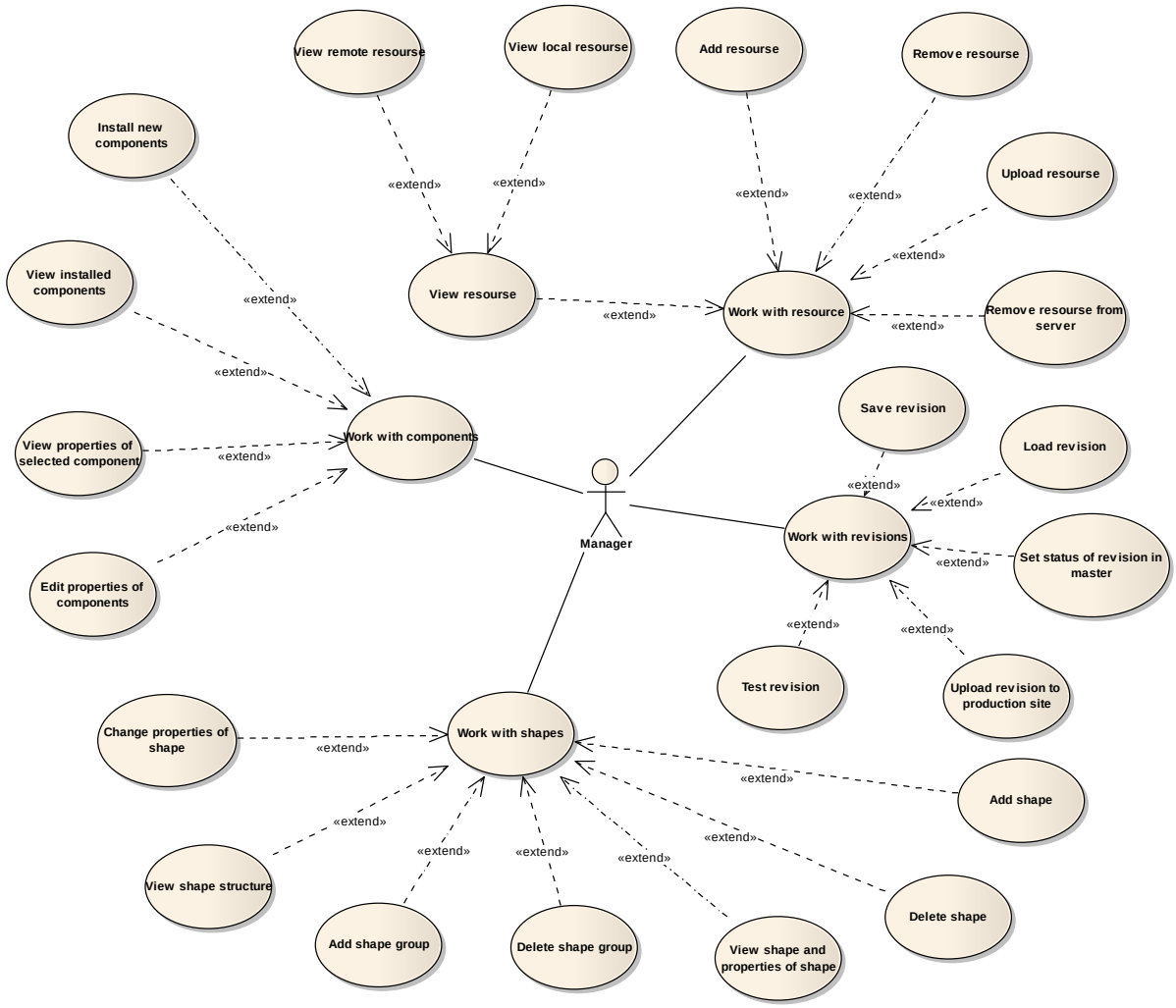

Figure 4. Use Case of the SCU

4.3 Manager.dll structure

In the Manager.dll library the actions related to resources, components, configuration information and scientific component deployment into PWA are implemented.

4.3.1 Main Manager.dll modules

The library structure is represented in Figure 5.

The library consists of the following modules:

1. Resources—module to manage resources. Provides operations such as to add new resources, or delete resources from server.
2. Components—module to manage scientific components. Provides operations such as add components, edit components, and delete component.
3. Configuration Information—module for working with component configuration information. Provides functions to add, edit, or delete component configuration information. It assigns components a specific configuration.
4. Revisions—the module to manage system revisions. Provides the creation of new revisions, deletion, and deployment of selected revisions to PWA production server.

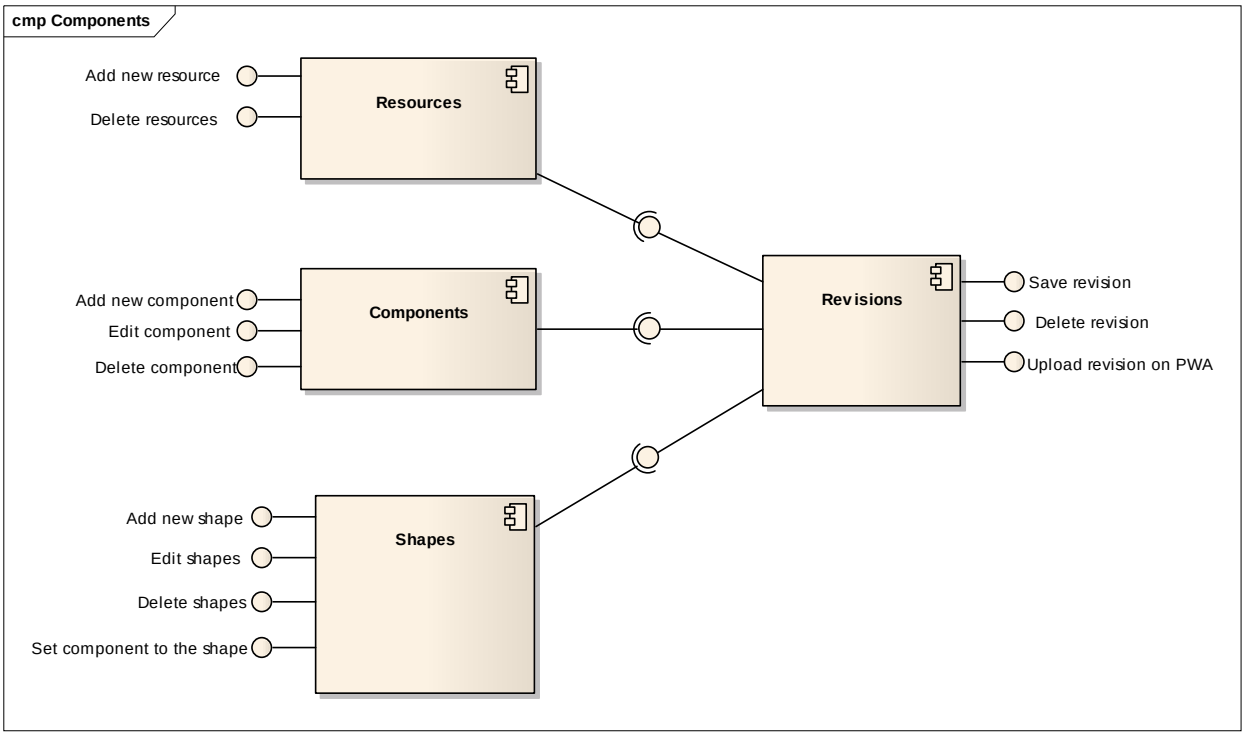

Figure 5. Manager.dll structure

4.3.2 Manager.dll classes diagram

The main Manager.dll classes diagram is shown in Figure 6.

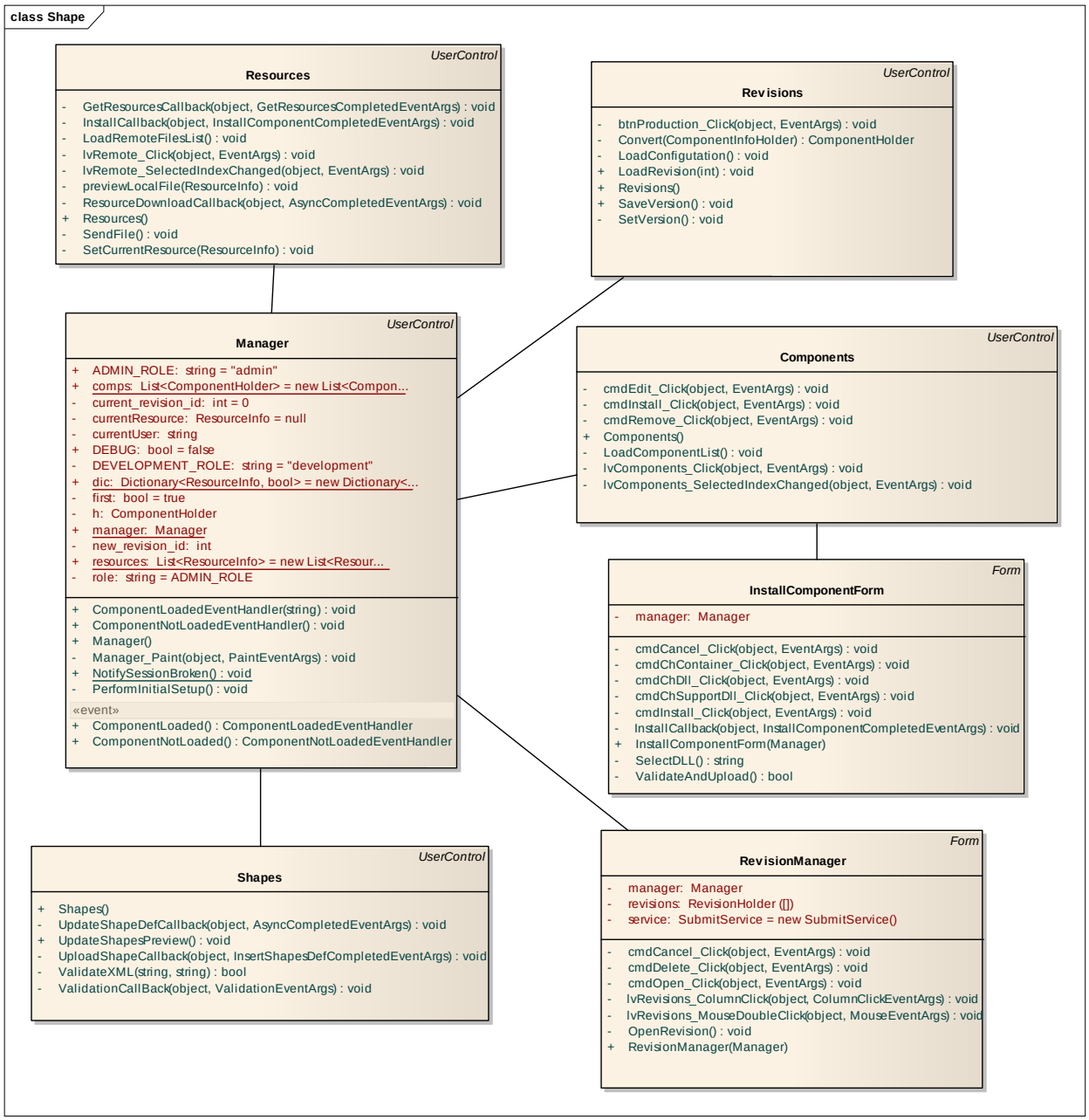

Figure 6. Manager.dll classes

Manager—this module implements main GUI library

Method	**Assignment**
Manager_Paint	The method activates at window repainting.
PerformInitial	Activates once at library download. In this method the revisions for current user are downloaded and the library download process is displayed.
NotifySessionBroken	Informs user about the exclusions, appeared during the process
componentLoaded	Component download

Resources—the work with resources is implemented. This class directly process user's activities

Method	**Assignment**
LoadRemoteFileList	The resource list download from server
SendFile	The file download to server
GetResourceCallBack	Response after the completion of resources download to server

InstallComponentsForm—GUI is implemented for work with components

Method	**Assignment**
ValidateandUpload	Checks the inputted data correctness and saves the components
cmdInstall_Click	The processor on request of component saving
cmdChDll_Click	The processor on request of library selecting for component
InstallComponentForm	New component addition

Components—work with components is implemented

Method	**Assignment**
LoadComponentList	Components list download
cmdInstall_Click	Opens the form for components information input
LvComponents_Click	The processors of component selecting in the list. Output the detailed information about the component

Revisions—the work with the revisions is implemented

Method	**Assignment**
btnProduction_Click	The processor on request of revisions download to PWA
LoadRevision	Downloads the specified revision
saveVersion	Revision saving

Shapes—the work with shapes is implemented

Method	**Assignment**
ValidateXml	Checks xml-description of shapes
UpdateShapesPreview	Shapes repainting after changing
UploadShapesCallBack	Shapes storing

RevisionManager—GUI for work with revisions

Method	**Assignment**
cmbDelete_Click	Revisions deletion
cmdOpen_Click	The opening of specified revision
lvRevision_Click	Selecting revision processor

4.4 Utility.dll structure

In the Utility.dll library user authorization and authentication management is implemented. Additionally the Utility.dll library implements server side functionality of the SCU by storing scientific components, configuration information, and resources in the database.

4.4.1 Main Utility.dll libraries

The library consists of the following modules:

1. *Authentication module*. Its main functions are to receive the user credentials, to get the users roles, and to start a new session for user
2. *Revision module*. The main functions of the revision module are to get a revision by id, delete a revision, identify the current revision, and add a new revision.
3. *Configuration Information Service*. The library provides such functions as get shapes on revision number, paste the new shape, and get shape by id.
4. *Component module*. The main functions of the component module are to get all of the scientific components, to get the scientific components by revision, to add a new scientific component, and to update the scientific component.

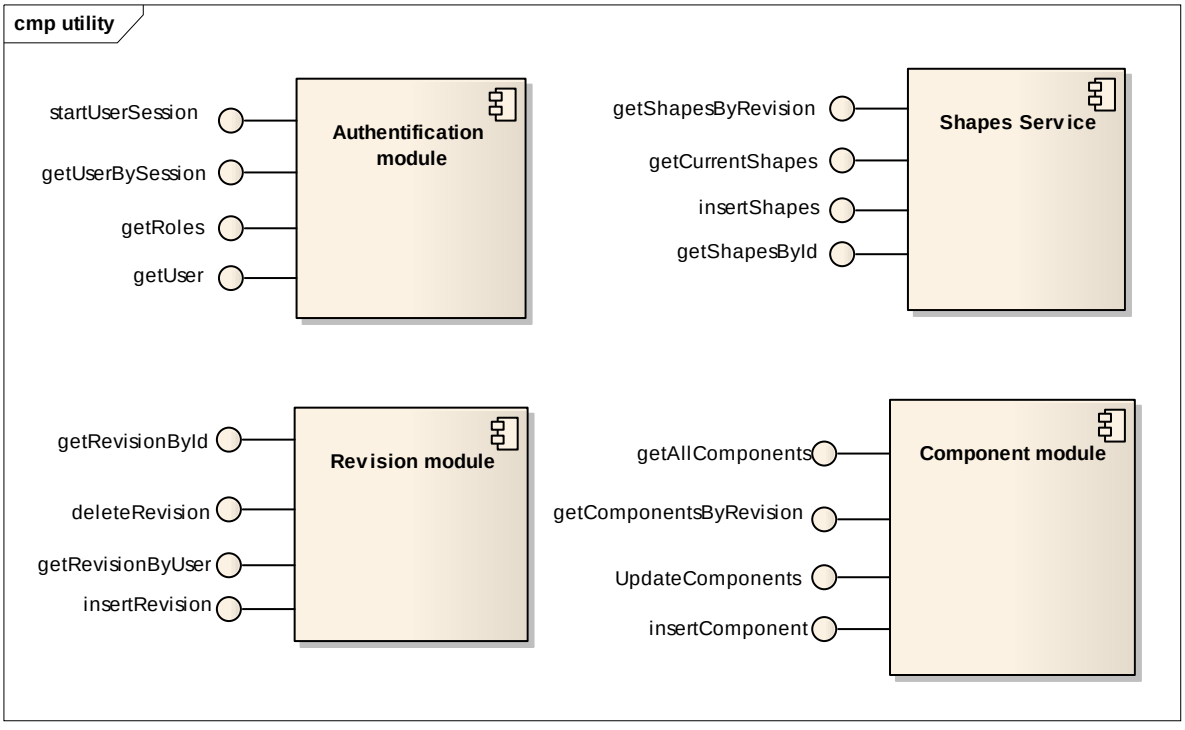

Figure 7. Utility.dll structure

4.4.2 Utility.dll classes diagram

The Utility.dll classes diagram is shown on Figure 8. The library consists of the following main classes.

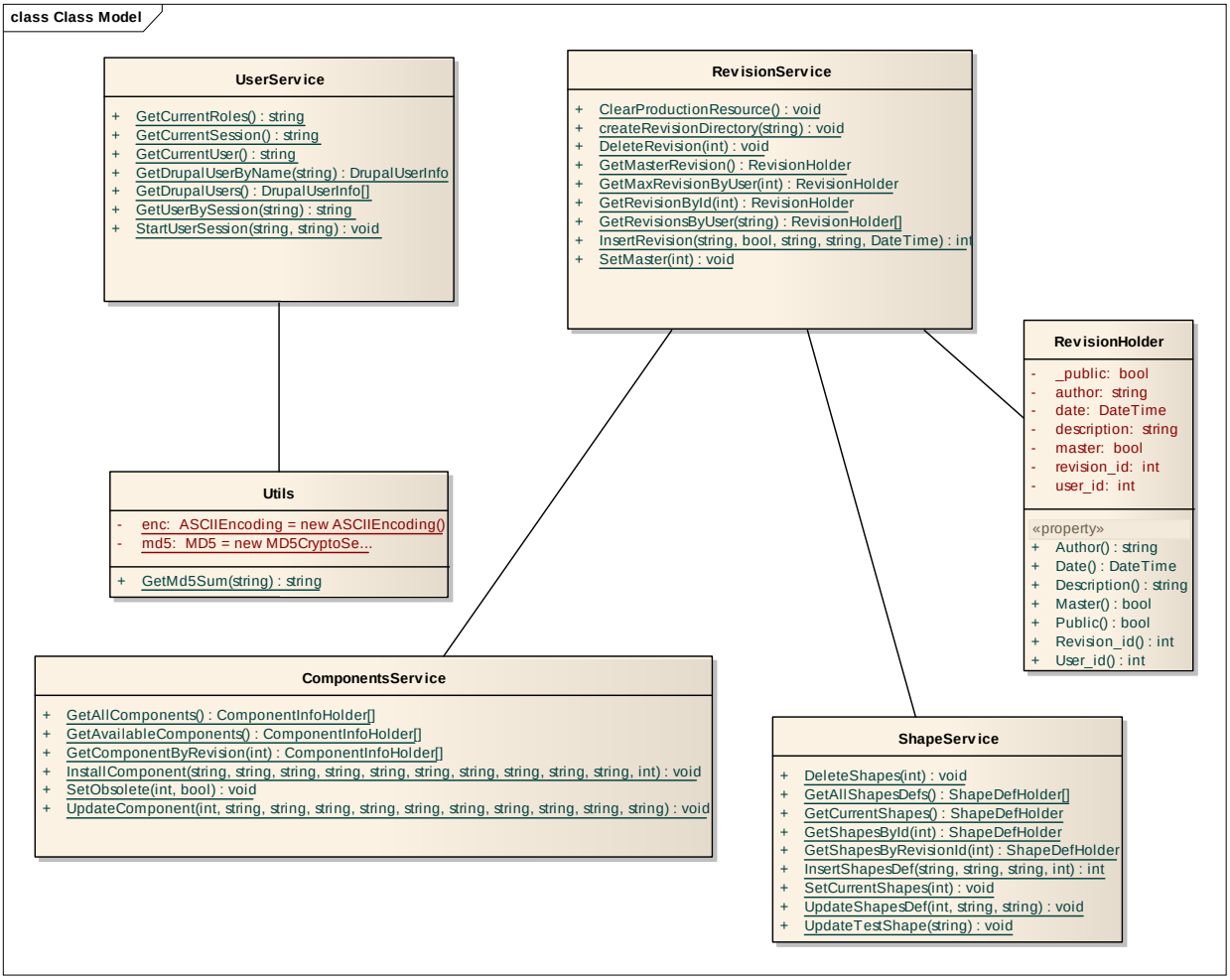

Figure 8. Utility.dll classes

UserService—in this class the service for work with user is implemented

Method	**Assignment**
GetCurrentRoles	Returns the current user roles
GetCurrentSession	Returns the session of current user
GetDrupalUsers	Returns all the users, which are registered in the system
GetUserBySession	Returns the users on session
StartUserSession	Starts the new session for specified user

RevisionService—service for work with revisions

Method	**Assignment**
CreateRevisionDirectory	Creates new catalog, where all the revisions and library component will be stored
DeleteRevision	Deletion of the specified revision
GetMaxRevisioByUser	Returns the last revision, which user edited
InsertRevision	Inserts new revision
GetRevisionById	Returns revision by specified id

ComponentService—service for work with components

Method	**Assignment**
GetAllComponents	Returns all the components
getComponentsByRevision	Returns the components by revision
UpdateComponents	Updates the information about the specified component
InstallComponent	Adds new component

ShapeService—service for work with shape

Method	**Assignment**
DeleteShapes	Deletion of specified shape
GetAllShapesDef	Returns all shapes
InsertShapesDef	Addition of new shape description
UpdateShapesDef	The update of specified shape
GetShapesByRevision	Returns the shapes description for specified revision
GetShapesById	Returns the shape description by specified id

4.5 Data Base structure

Data base structure is shown in Figure 9.

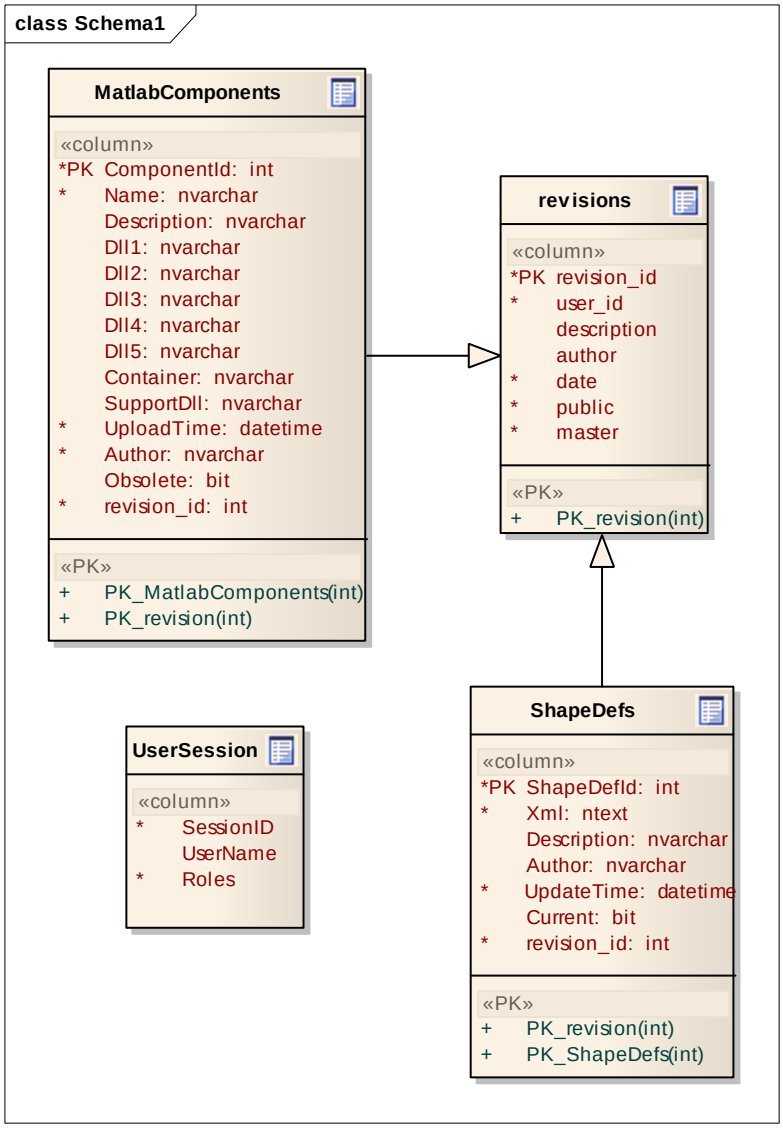

Figure 9. Data base structure

UserSession—A unique line identifying the user's session is stored in this table

Method	**Description**
SessionId	Session identifier
UserName	User's login
Roles	User's roles

Revisions—the revisions created by users are stored

Method	Description
Revision_id	Revision identifier
User_id	User identifier, who stored this revision
Description	Revision description
Date	Creation date
Public	The indication that the revision is public and available for viewing for all the users
Master	The indication that the revision has already been deployed to the PWA

ShapeDefs—shapes description

Method	Description
ShapeDefId	Shapes description Identifier
Xml	Xml-description shapes
Description	Shapes description
Author	The author who created shapes
updateTime	The update time
Revision_id	The revision identifier, to which the shapes refer

MATLABComponents—component information

Method	Description
ComponentId	Component identifier
Name	Component name
Description	Component description
Dll1, Dll2, Dll3, Dll4, Dll5	Libraries names
SupportDll	Library name, that provides the interface of connection with PWA
Revision_id	Revision identifier, to which the component is referring
Author	The author who created the component
Obsolete	The indication of that the component is out of date
UploadTime	The component download time

4.6 Web-services

Web services provide a convenient method of communication between client and server via HTTP protocol. A short description of main methods is presented below.

Method	Description
GetAllComponents	Returns all the components from server
GetShapesById	Returns shapes by id
GetResources	Returns resources by revision
GetLastRevisionToUser	Returns the last revision for user
InstallComponent	Adds new components
InsertShapesDef	Adds new shapes
GetShapesByRevision	Returns shapes by resource identifier
GetComponentByRevision	Returns components by revision identifier

5 PrIMe Workflow Application

The purpose of the PrIMe Workflow Application (PWA) is to provide a user interface for working with scientific workflow projects.Using the PWA, the user can create, open, and execute scientific workflow projects. A scientific workflow project is comprised of a network of linked scientific components.

5.1 System architecture

The general system architecture of the PWA is represented in Figure 10.

The PWA consists of the following elements:

1. *Server*. The server is the main server where the PWA is executed from. All of the functions of the server are accessed through the library Utility.dll. The authorization and authentication process and also the database work are accomplished by means of the Utility.dll library. The interaction with the client's browser and application servers is facilitated through web-services.
2. *Application server*. Remote applications are executed remotely from the client on the application servers. Web-services facilitate interaction with the client's browser and PWA.
3. *Client*. PWA is executed in the context of the client's browser with the use of the PrIMeKineticsClient.dll library. Through the client the user can create scientific workflow projects, update existing scientific workflow projects, and execute scientific workflow projects. The PrIMeKineticsClient.dll library interacts with MATLAB components through a library called ComponentsFromMatLab.dll, with which it directly communicates.
4. *MATLAB components*. The MATLAB components are downloaded to the client and activated at the execution of the scientific workflow project.

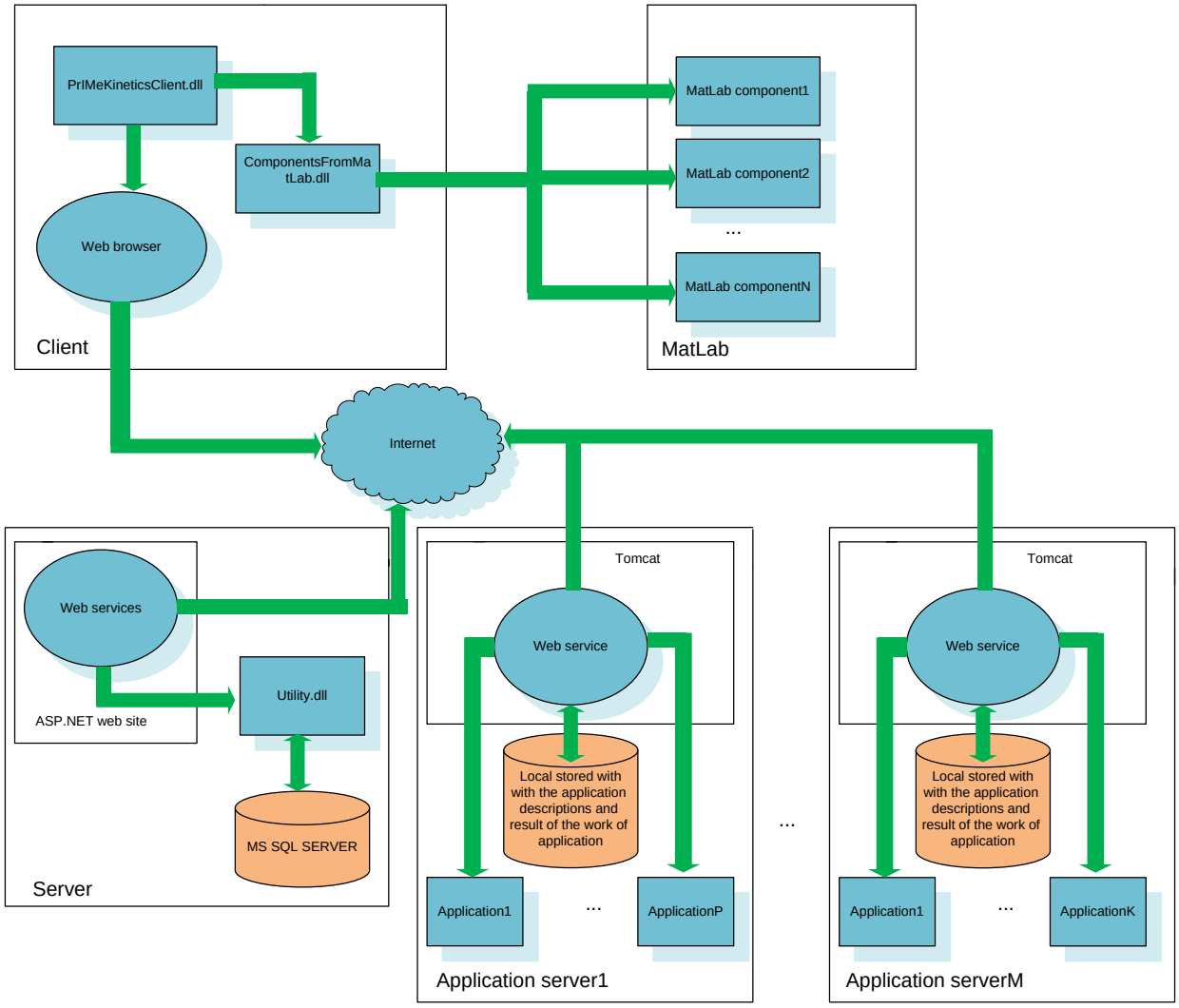

Figure 10. PWA Architecture

5.2 Use Case PWA

The function of the PWA is to work with scientific workflow projects. Use Case of the PWA is represented in Figure 11.

The main functions of the PWA are:

1. Project management. The PWA enables creating, editing, and deleting of scientific workflow projects.
2. Project collaboration. The PWA allows a user to classify a scientific workflow project as private, shared, or public. Setting a scientific workflow project as shared or public allows multiple users to collaborate on a project.
3. Custom project building. The PWA allows the user to create a scientific workflow project from available scientific components. The user accomplishes this by moving scientific

components to the project plane, defining the scientific component relationships with links, and specifying scientific component inputs and properties.

4. Project execution. Once the scientific workflow project is created, the scientific components linked, and the inputs and properties are set for each scientific component the project can be executed in PWA. Following execution, the results can be viewed in PWA.

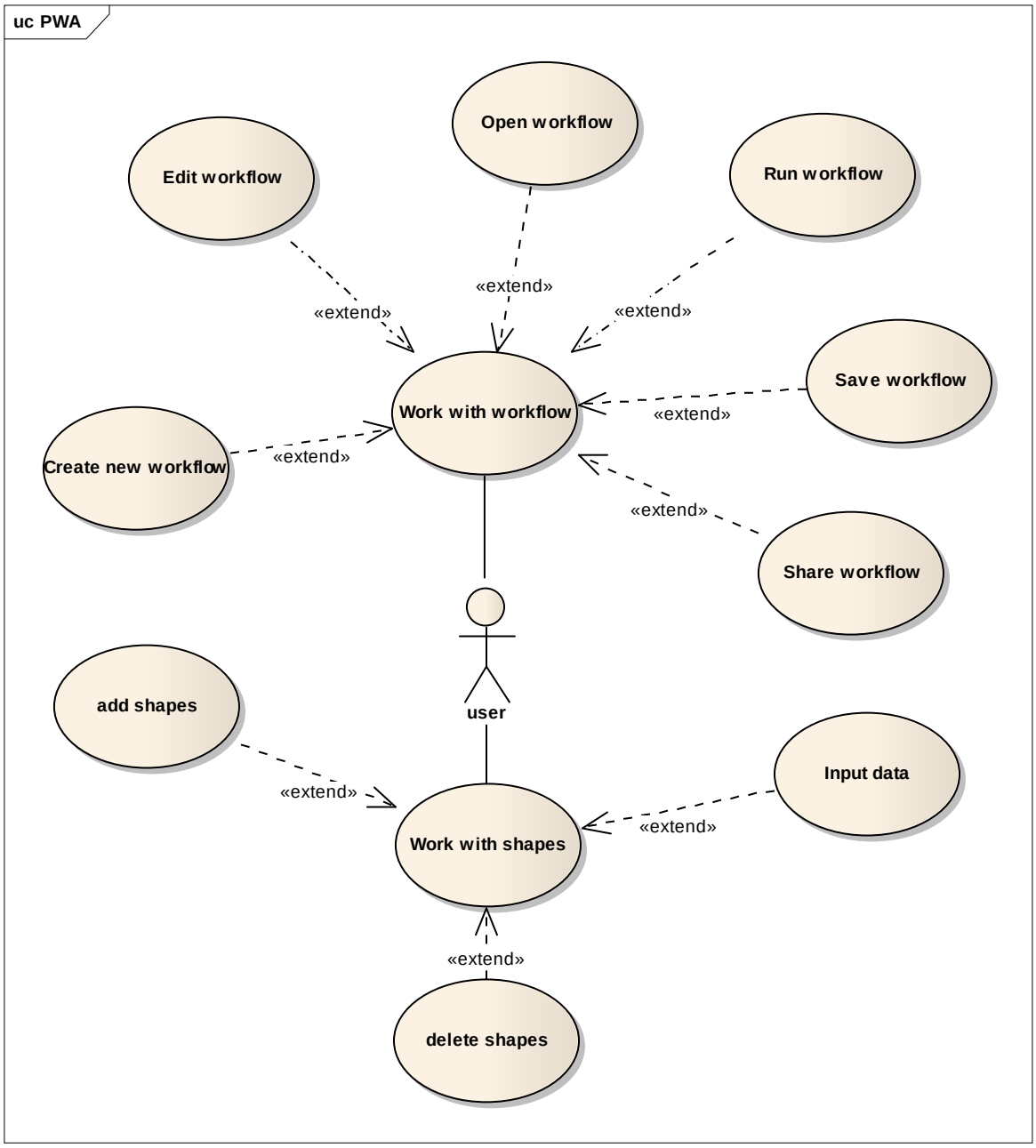

Figure 11. Use Case diagram of the PWA

5.3 Component types

There are two types of scientific components available for building of scientific workflow projects: local and remote. The following is a description of each type.

1. *Local components*. The local components, created using MATLAB, are executed directly from the user's computer. Upon execution of a scientific workflow project local components are copied to the client computer. The work results from each local component are also saved on the client computer.

2. *Remote components*. Remote components are executed from remote servers and interact with PWA using web-services. Multiple remote components can be hosted on a single server.

5.4 The component integration with PWA

5.4.1 Local components (MATLAB)

In order for the PWA to execute the local MATLAB components on user's computer the MATLAB runtime library must be installed.

A special support library, called ComponentsFromMatLab.dll provides the interface between the client browser and the local scientific components. PrIMeKineticsClient.dll communicates with the local scientific components through the ComponentsFromMatLab.dll library. Upon execution of a scientific workflow project each component is executed by means of the library methods which control it. At the beginning of project execution the support library downloads each MATLAB component to the local computer. Each component is executed by the MATLAB runtime library. The support library, ComponentsFromMatLab.dll, controls the execution of each local scientific component.

When the scientific workflow project is executed all of the project information is stored on client's computer in a catalog called ProjectData.xml. This catalog encodes the scientific component relationships, description, properties, the location of results, and the completion status of each component. The paths of each component results are given in the catalog. ProjectData.xml is recorded only once at scientific workflow project execution. Below, an xml example is represented. Each component in the project is represented by a <node> element, in which all of the input properties and information about the connected components is recorded. The completion status of each component is recorded as 1 (success) or 0 (failure).

```
<?xml version="1.0" encoding="utf-8" ?>
<project id="165" modified="29.09.2008 15:26:55" executed="" creator="alx" status="1" lastInternalId="8" >
  <nodes>
    <node id="2" name="Model 1" group="Models" type="Model" >
      <properties>
        <status>0</status>
        <resultObj></resultObj>
        <about>This node supplies a model.</about>
        <icon>model.gif</icon>
        <list description="Model Source" group="attributes" name="Source" readOnly="False">
          <option value="1" caption="PrIMe Warehouse" link="http://prime-
          warehouse.berkeley.edu/depository/models/catalog/m00000003.xml" selected="true" >from PrIMe Warehouse</option>
          <option value="2" caption="local" link="" >from local machine</option>
        </list>
        <list description="Model Type" group="attributes" name="Type" readOnly="False">
          <option value="1" caption="detailed" link="" selected="true" >A detailed model</option>
```

```
            <option value="2" caption="reduced" link="" >reduced model</option>
            <option value="3" caption="tabulated" link="" >tabulated model</option>
          </list>
        </properties>
        <location width="92" height="62" x="43" y="315" />
        <layout>
          <image x="0" y="0" file="model.gif" />
        </layout>
        <inputs>
        </inputs>
        <outputs>
          <output id="1" x="45" y="54" linkNodeId="4"
           linkNodeInputId="1">c:/PrIMe_Workflow/projects/project_165/nodes/node_2/grimech30.mat</output>
        </outputs>
      </node>
    </nodes>
</project>
```

We will now review the ComponentFromMatLab.dll in more detail. The purpose of the ComponentsFromMatLab.dll library is to provide methods for scientific components developed by third-party developers, such as custom built MATLAB programs, to be integrated with PWA. Scientific components must follow integration rules defined in PWA (ProjectData.xml). The interface library must contain one or more classes with methods having the following prototype:

```
public static void RunComponent(string componentPath, string integrationCatalogPath, string worflowId, string
  nodeId)
```

Each of the arguments is described below:

componentPath—absolute path of component files location
integrationCatalogPath—absolute path of integration files location
worflowId—workflow ID
nodeId—node ID

Below is an example of a .NET interface for local MATLAB components:

```
using System;
using System.Collections.Generic;
using System.Text;
public class MatlabCompsSupportClass
{
   public static void RunPFR(string ctfPath, string path, string wfId, string nodeId)
   {
      MatlabComponents mc = new MatlabComponents(ctfPath); // instantiating main class for components bundle
      mc.run_pfr(path, wfId, nodeId); // running the component
   }
}
```

It is necessary to modify the main stub class generated by MATLAB Builder for .NET in order to pass the CTF file location explicitly because it uses current directory by default. The only requirement is to replace the static constructor with a constructor having the installation path as a parameter. An example is shown below.

```
public MatlabComponents(string ctfFilePath)
   {
      if (MWArray.MCRAppInitialized && mcr == null)
      {
         mcr = new MWMCR(MCRComponentState.MCC_matlab_comps_name_data,
                  MCRComponentState.MCC_matlab_comps_root_data,
                  MCRComponentState.MCC_matlab_comps_public_data,
                  MCRComponentState.MCC_matlab_comps_session_data,
```

```
                MCRComponentState.MCC_matlab_comps_matlabpath_data,
                MCRComponentState.MCC_matlab_comps_classpath_data,
                MCRComponentState.MCC_matlab_comps_libpath_data,
                MCRComponentState.MCC_matlab_comps_mcr_application_options,
                MCRComponentState.MCC_matlab_comps_mcr_runtime_options,
                MCRComponentState.MCC_matlab_comps_mcr_pref_dir,
                MCRComponentState.MCC_matlab_comps_set_warning_state,
                ctfFilePath, true); // pass contuctor parameter to MWCR initializer
        }
        else
        {
            throw new ApplicationException("MWArray assembly could not be initialized");
        }
    }
```

5.4.2 Remote components

Configuration information of the remote components must have the ability to add “remote execution” tag to a component. This tag will have following properties:

- Application name
- IP address and port of remote server

The PWA server stores information for each scientific workflow project which contains remote components such as projectid, nodeid, applicationid, status, and jobid. For each executed scientific workflow project a unique jobid is assigned.

There is a button on the scientific component property page which runs a remote application. When user clicks it the following will happen:

1. Workflow process client makes request to main server.
2. The PWA server checks the status of this node. If it’s not running the server makes a web-service call to the appropriate remote server. The component job status is switched to “running” and project status becomes “read-only” until the remote application finishes the job or fails. Only one remote process can be run for each project at a given time.
3. The PWA user can check status of remote execution.
4. The PWA server polls the status of each component. Any delay is recorded in a config file. When the job is finished the main server downloads results and sets job status to “competed” and project status to “editable”.

According to the multiplatform implementation requirement we propose that a remote server communication module in Java is implemented. This module will have following functions available via web-services:

- ping—to check connection and server availability
- add job parameters
- start job
- check job status
- return job results

5.5 Executing Project

After the creation of the scientific workflow project, the user can execute it. The scientific workflow can consist either of local components (MATLAB), or of remote components, which are implemented on remote application servers. The execution logic is the same for remote and local components. In Figure 12 the process of executing a scientific workflow project is shown.

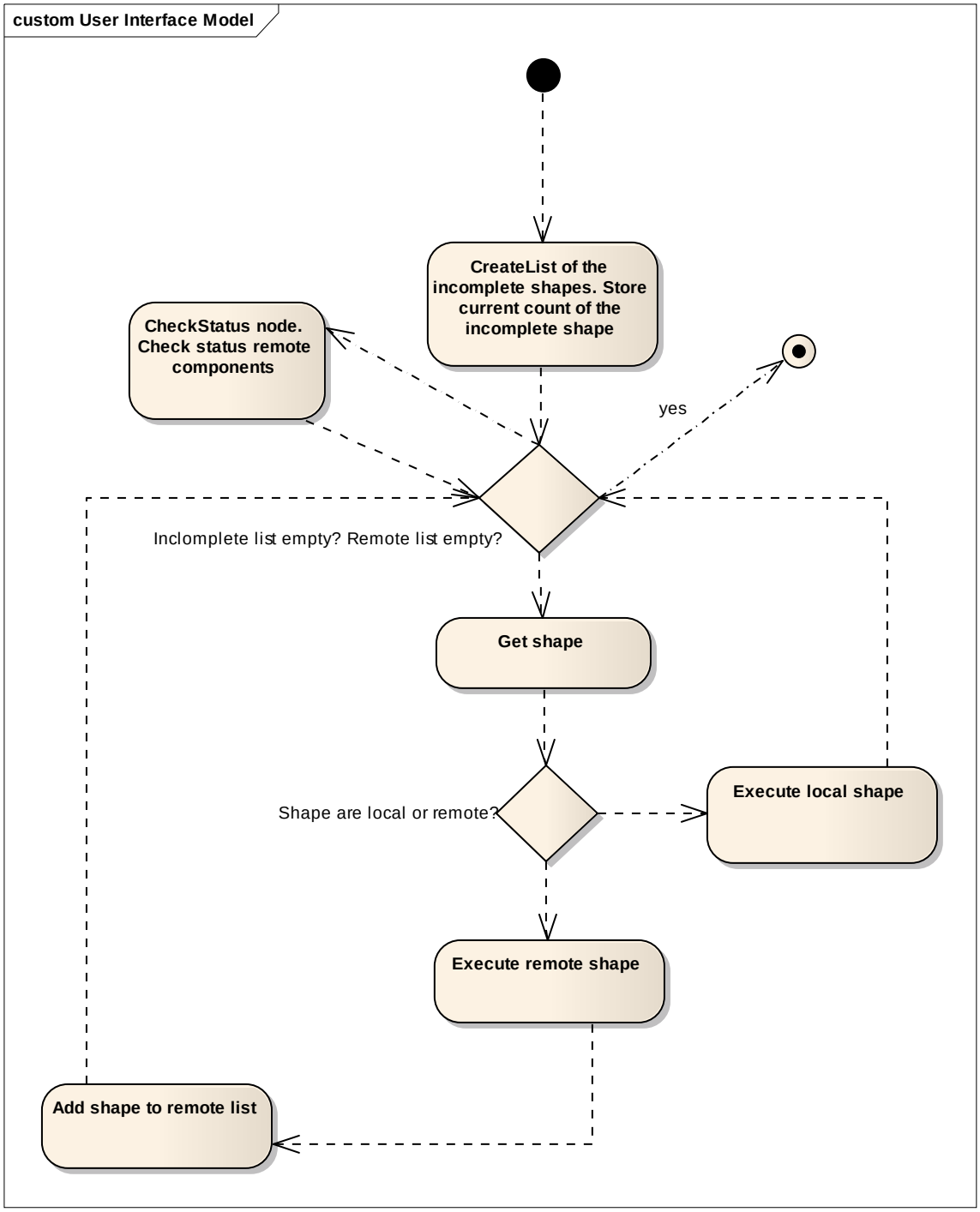

Figure 12. Activity diagram of executing a scientific workflow project

The process to start execution of local components is trivial. The appropriate method from the support library is simply activated. The execution of a remote component is represented more exactly in Figures 13 and 14.

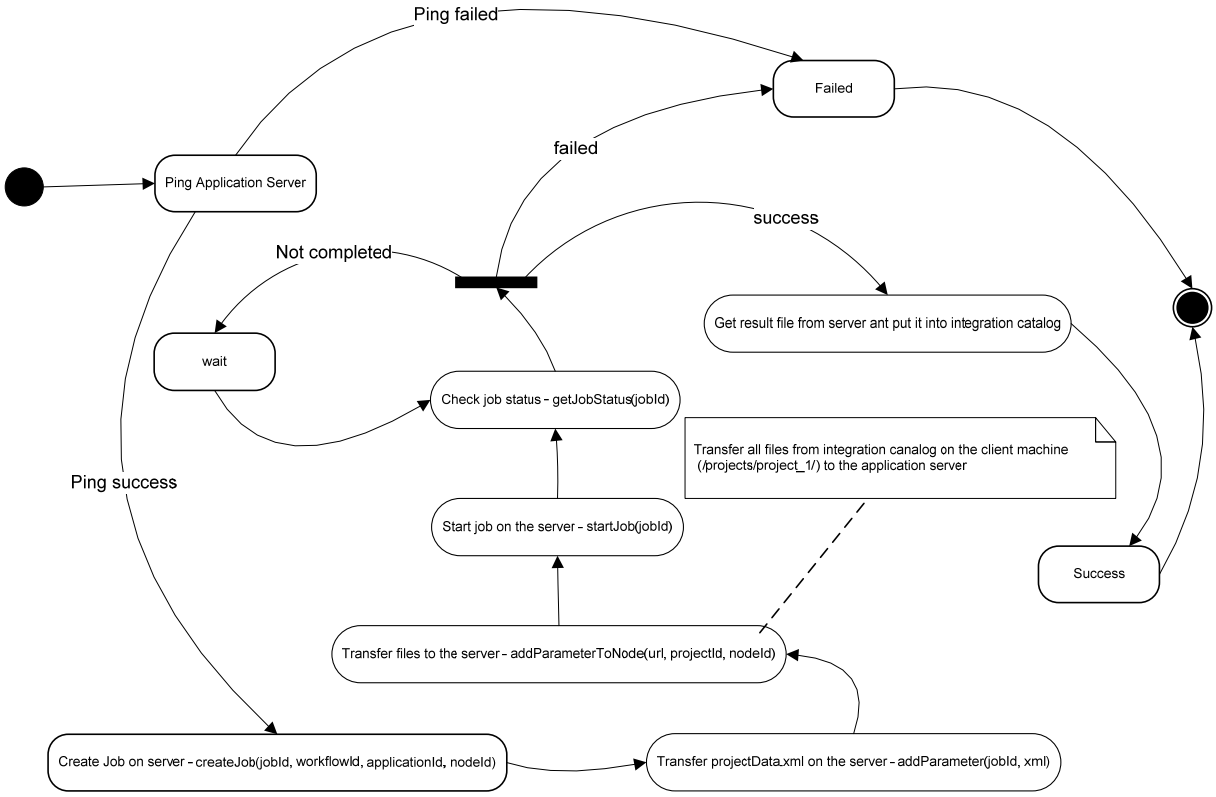

Figure 13. State diagram of execution of the remote component

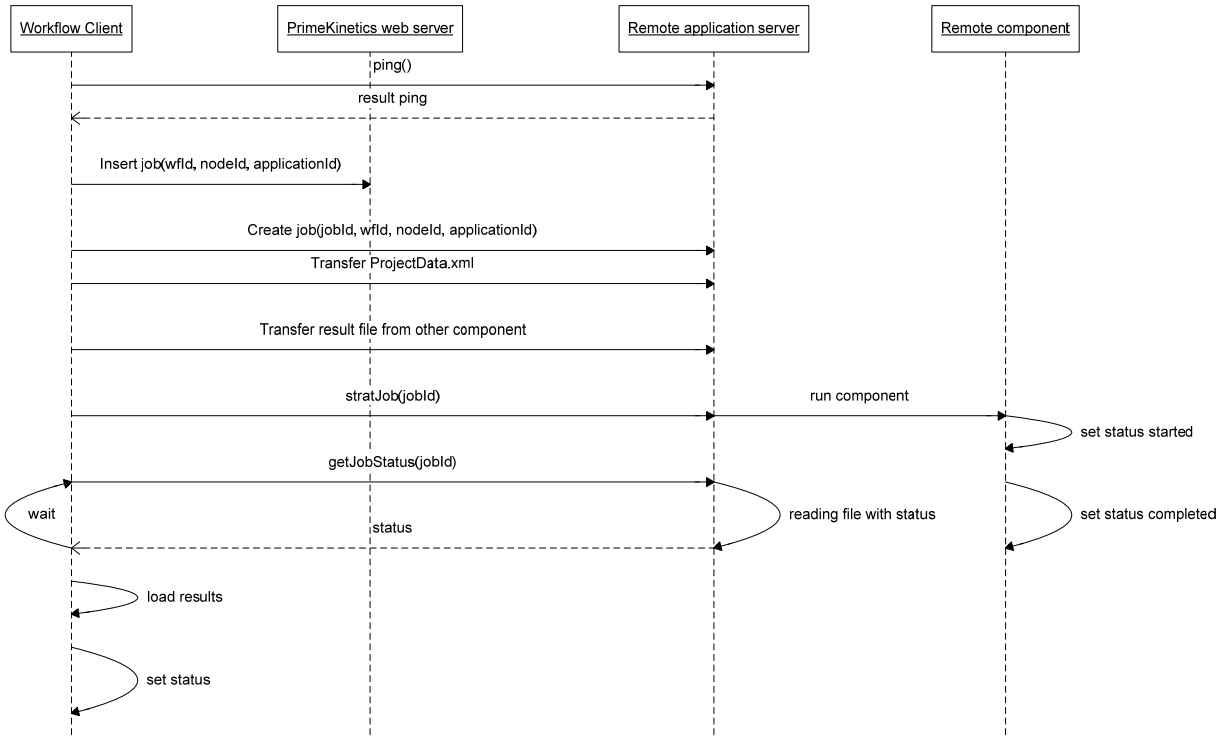

Figure 14. Activity diagram of executing a remote component

5.6 User's computer

Upon starting PWA, the following three libraries are copied to the client computer: PrimMeKineticsClient.dll, ComponentsFromMatLab.dll, and matlab_component.dll. All functions for working with diagrams, starting project implementation, and interaction with server applications are executed in these libraries. Figure 15 shows all of the elements of PWA that are stored on the client computer.

PrIMeKineticsClient.dll—This library is implemented in the context of browser, in which all the functions for creating and editing scientific workflow projects and their execution is implemented.

Support Library (ComponentFromMatLab.dll)—This library represents the interface for the interaction with MATLAB components. PrIMeKineticsClient.dll does not know about the components and their structure, yet it only knows the methods of the support library by which they are controlled.

MATLAB component(matlab_comps.dll)—This is the library, which directly controls the programs from MATLAB. This library is generated by the MATLAB Builder for .NET application. The library is connected to the support library and represents the required classes and functions for MATLAB programs.

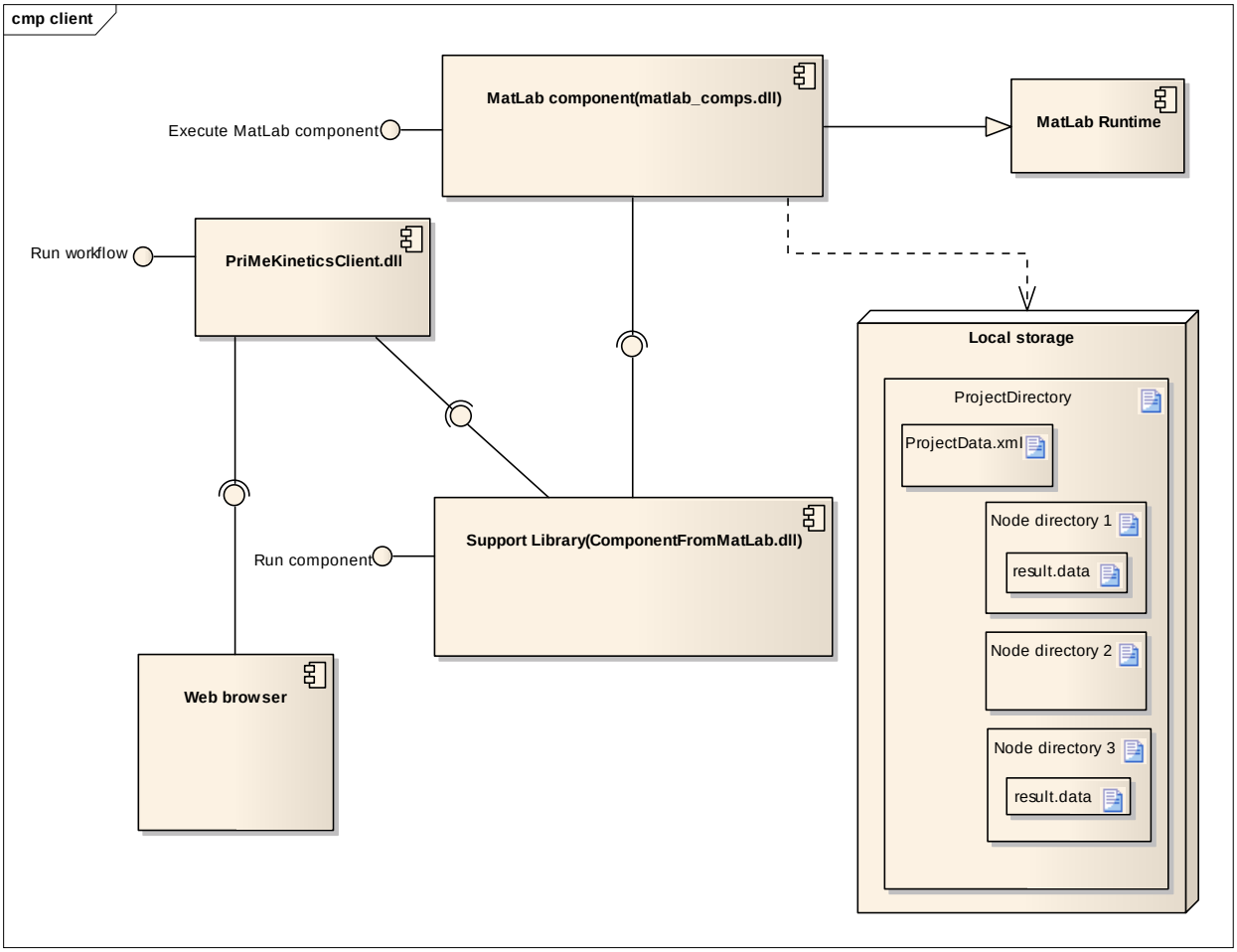

Figure 15. System components which are located on the user's computer

5.6.1 Main modules of the PrIMeKineticsClient.dll library

The general structure of the PrIMeKineticsClient.dll library is represented in Figure 16. It consists of the following parts:

1. *Workflows*. This module provides the methods that control scientific workflow projects such as creation, opening, and deletion of projects, and managing of the system users' access to the existing projects.

2. *Shapes*. In this module the graphical editor is controlled. It includes how components are displayed, the properties and input data of each component, and the relationships of each component in a scientific workflow project.

3. *Execute workflow*. This module controls how each component is executed and how the scientific workflow project is executed as a whole. This module manages the process and order of execution of every component and identifies whether the component is local or remote type.

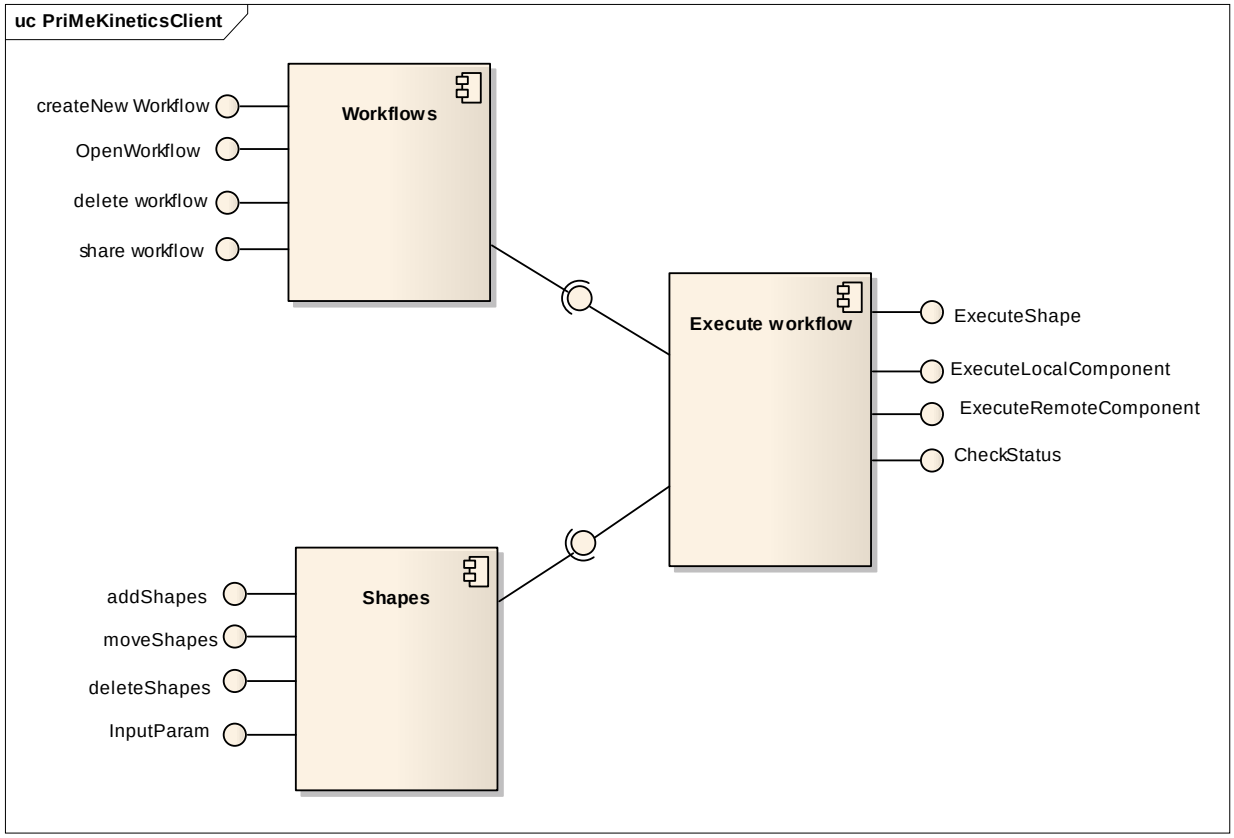

Figure 16. General PrIMeKineticsClient.dll structure

5.6.2 Classes diagram

The Classes diagrams for the PrIMeKineticsClient.dll library are presented in Figures 17 and 18. Below a description of each method and class assignment is presented.

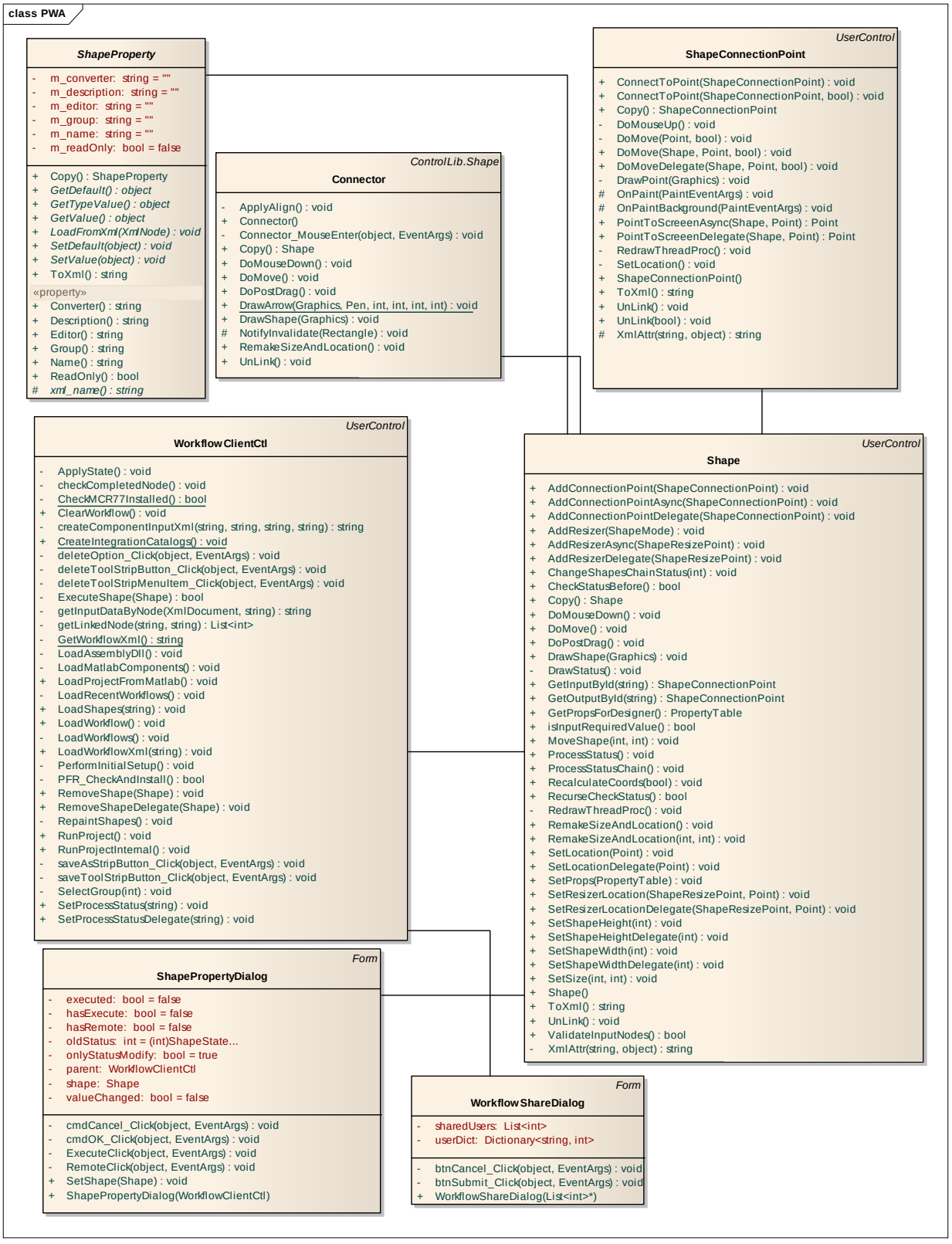

Figure 17. PrIMeKineticsClient.dll classes diagram

Shape—the components element, its subtle portrayal, properties set and editing are controlled by this class

Method	Description
AddConnectionPoint	Adds entry or exit to shape
ChangeShapesChainStatus	Changing shape status at editing of its connections with other shape.
Copy	Shape copy creation. Is activated when user drags a new shape on the diagram
DoMouseDown	Processes the user's mouse clicking on shape
isInputRequiredValue	Checks whether the field is required for execution
MoveShape	Shape navigation processing
SetLocation	Shape setting in specified position on screen
SetProps	Property establishment, which user entered
ToXml	Shape conversation and all its properties in xml
UnLink	Is activating at connection deleting with other elements on diagram
ValidateInputNodes	Checks the correctness of entries and exits
XmlAttr	Returns by name the attribute value
GetInputById	Returns the shapes entry by identifier
GetInputById	Returns shapes exit by identifier
SetSize	Shape size set
RedrawThreadProc	Is activating in separate flow for shape subtle portrayal at navigation

ShapeConnectionPoint—is used for the connecting components displayed in the scientific workflow project

Method	Description
ConnectToPoint	Connects the set point with the other
SetLocation	The position set on shape
DrawPoint	Point subtle portrayal on the screen
ToXml	All the point conversion in xml
UnLink	Connection deletion of the point set

Connector—is used for displaying lines which connect two diagram elements

Method	Description
doMouseDown	Processes the user left mouse click and starts to draw the connecting line
doMove	The process of navigation, while the connecting line changes its size, depending upon the cursor position on the screen
remakeSizeAndLocation	Size and position changing at shapes navigation
UnLink	Diagram deletion
ApplyAlign	The application of changes set by user

ShapeProperty—is used to set shape properties

Method	Description
LoadFromXml	Property download from xml presentation
SetValue	Property setting
ToXml	Property conversion in xml presentation
GetValue	Returns the property value

WorkFlowClientCtl—used by main GUI library

Method	Description
ApplyState	A new condition application to diagram
CheckCompletedNode	Diagram nodes checking
CheckMCR77Installes	Checking whether the MATLAB Runtime is installed
ClearWorkflows	New workflow creation
ExecuteShape	Component start for application, which is connected with the indicated shape
getInputDataByNode	Returns the shape properties, inputted by user
LoadAssemblyDll	Client component download
LoadProjectFromMatLab	Xml download, which modified MATLAB at its components application

LoadWorkflow	Workflow opening
RunProject	The user request: “to start the project” processing
RepaintShapes	All shapes repainting on diagram
RunProjectInternal	Is running in separate flow and manages the diagram starting process
SetProcessStatus	Displays the diagram starting progress
SaveAsStripButton_Click	Project storing
getLinkedNode	Returns all the shapes, which are connected on the diagram with the set
CreateIntagrationCatalog	Creates the catalogue on the user’s computer, to which work results will be stored
PerformInitialSetup	Is activating at download, makes all the necessary initialization, display the library download process in client’s browser
RemoveShapes	Shape deletion from the diagram
LoadRecentWorkflows	The download of available for user diagrams

WorkflowShareDialog—GUI to provide the specified projects access to users

Method	**Description**
btnSubmitClick	The application of rights, set by user
btnCancel_Click	The processing of user clicking cancel
workflowShareDialog	Kit, downloads and displays the list of all the system users

ShapePropertyDialog—GUI setting shape property

Method	**Description**
cmdCancel_click	The processing of user clicking cancel
cmdOK_Click	The application of all the properties set by user
SetShape	Connects the selected shape from the GUI data, so displays the inputted properties and remembers shape, for storing of new properties

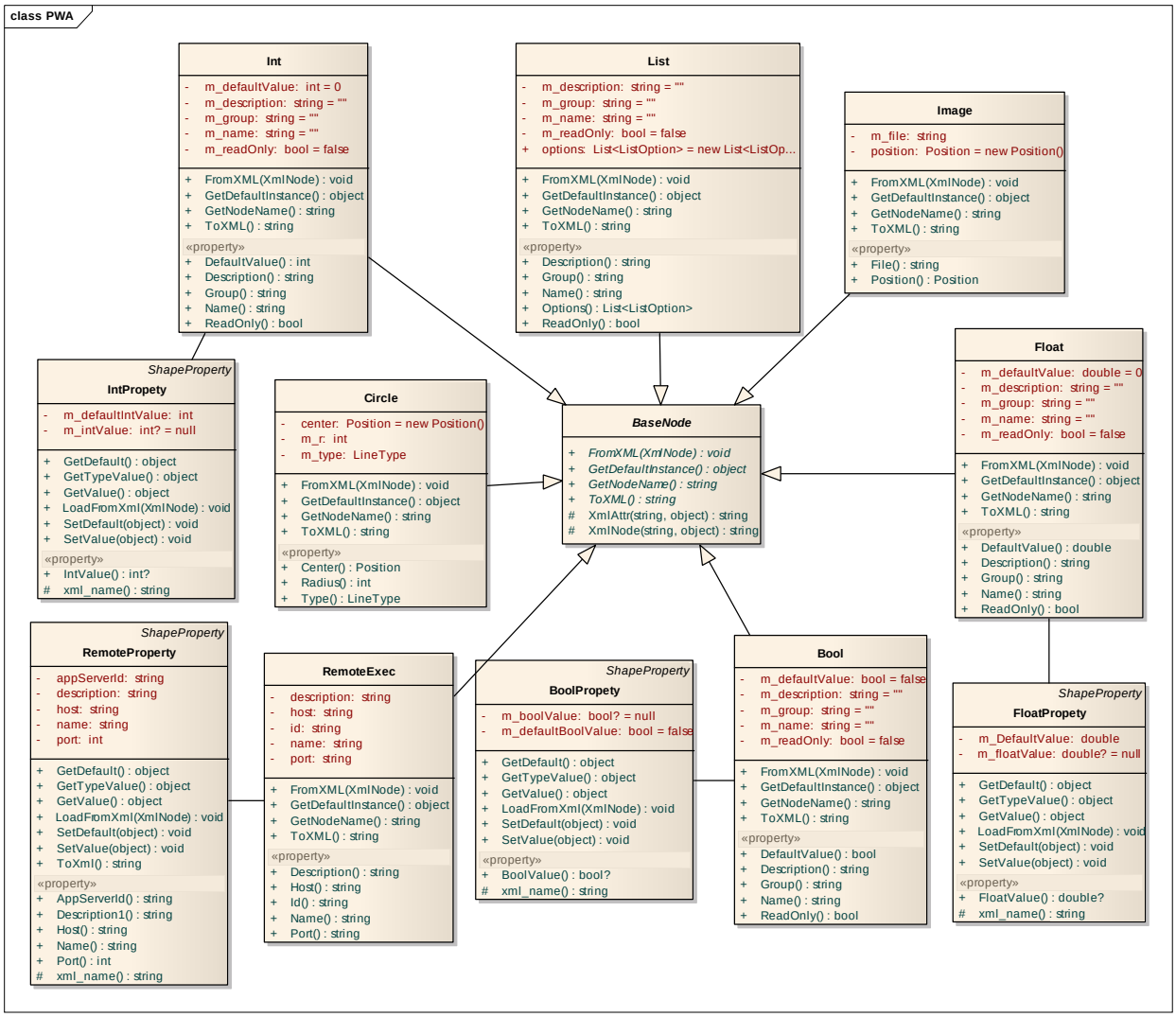

Figure 18. PrIMeKineticsClient.dll class diagram

Classes displayed in Figure 18 are used for the storing information about the scientific workflow projects and their structure stored in ProjectData.xml.

BaseNode—is used for the storing information ProjectData.xml. This is the base class

Method	**Description**
FromXml	Creates the node on the bases of xml description
GetNodeName	Returns the node name
XmlAttr	Returns the attribute by indicated name
XmlNode	Creates XmlNode by indicated information

All the other classes Bool, Int, Float, List, RemoteExec—descend the BaseNode class. Their methods are trivial which is why we will limit it to the BaseNode description.

5.7 Application server structure

The Application server stores the applications which are implemented remotely from client's computer. The interaction with the application server is accomplished through web-services. The main components of the application server are discussed below.

Web-service—on the Application server web-service is installed, which provides the interface for the interaction with PWA. Web-service also allows input parameters, component properties, and component results to be communicated between the application server and the main server.

Application—application, that makes the necessary calculations based on the input parameters. Each application must correspond to defined requirements, which are described in the following paragraph.

Local storage(config files)—these are the necessary configuration files, used by the web-application.

1. Application.xml—file, where the applications register and the path of the execution file is indicated.
2. Jobs.property—property-file where the information about the current tasks is stored.
3. Config.properties—configuration profile where the indicated path to the catalog of application results is saved. This file also creates a log and stores the path of the application.xml and jobs.property files.

ProjectDirectory—for each project a project directory is created which stores the project information in a ProjectData.xml file. Each project node has its own directory where the specific node results are stored.

5.7.1 Application server structure

In Figures 19 and 20 the structure and modules of the Application server are shown. Web-service facilitates the interaction of the application server with PWA, and manages the application starting process.

The main modules are presented below.

1. *WebService*. Web-methods are set, which are used for the interaction with PWA, and managing the process of starting the application on the Application server. ComponentRun. Controls the execution of the scientific application.
2. *Config*. Provides access to the main configuration files.

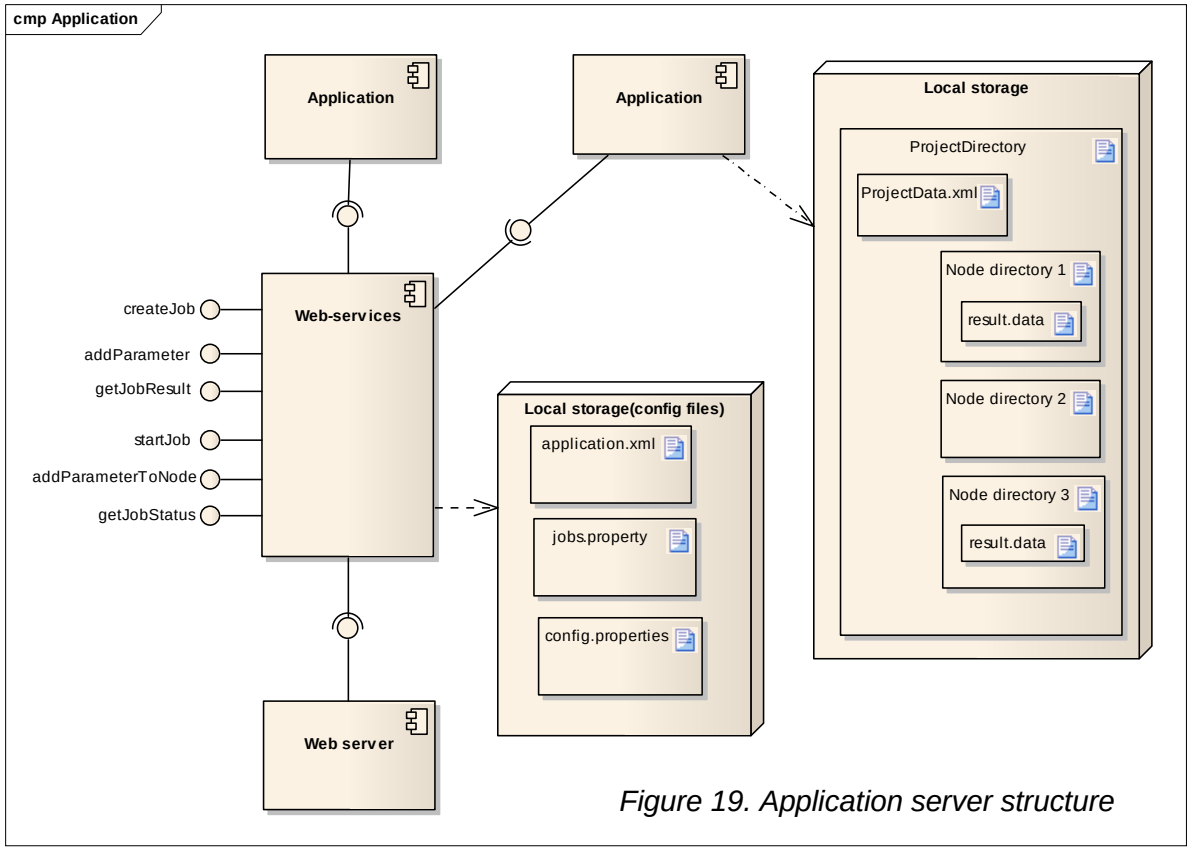

Figure 19. Application server structure

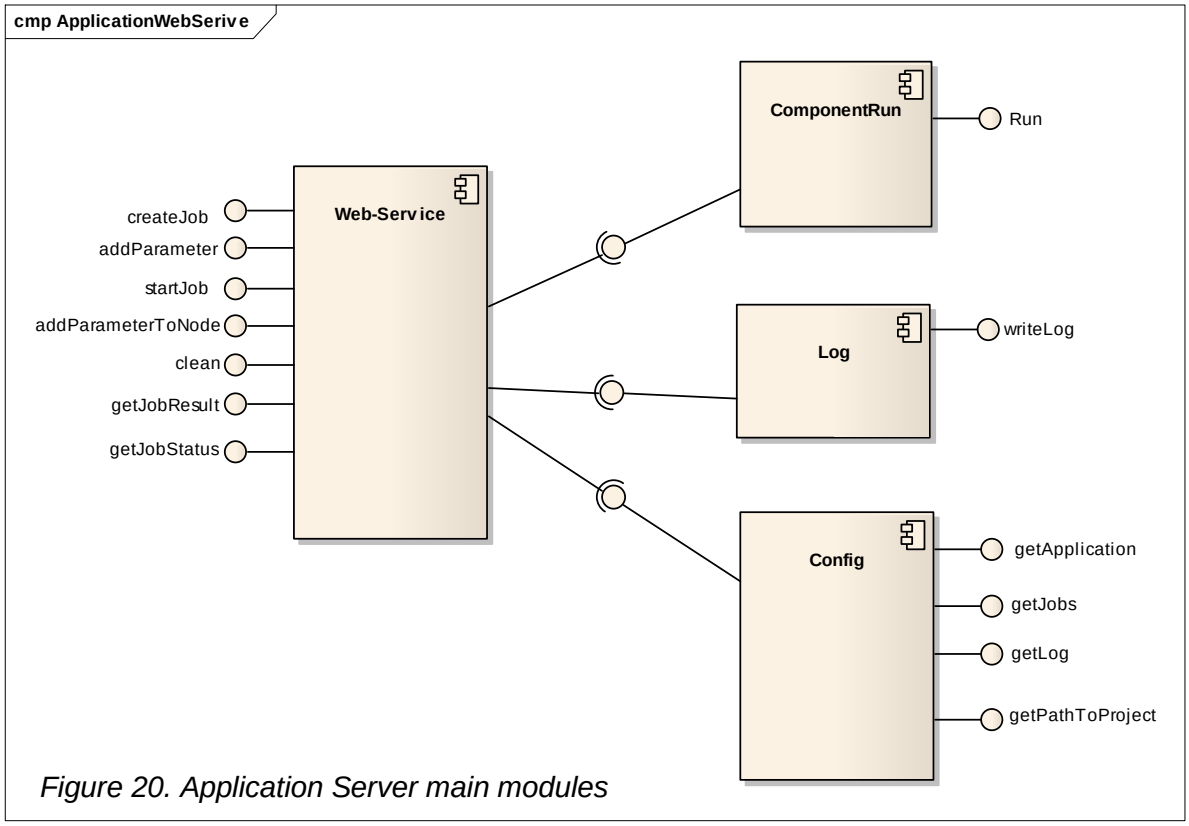

Figure 20. Application Server main modules

5.7.2 Classes diagram

The main classes of the Application server are represented in Figure 21. A description of the Application server and classes follows.

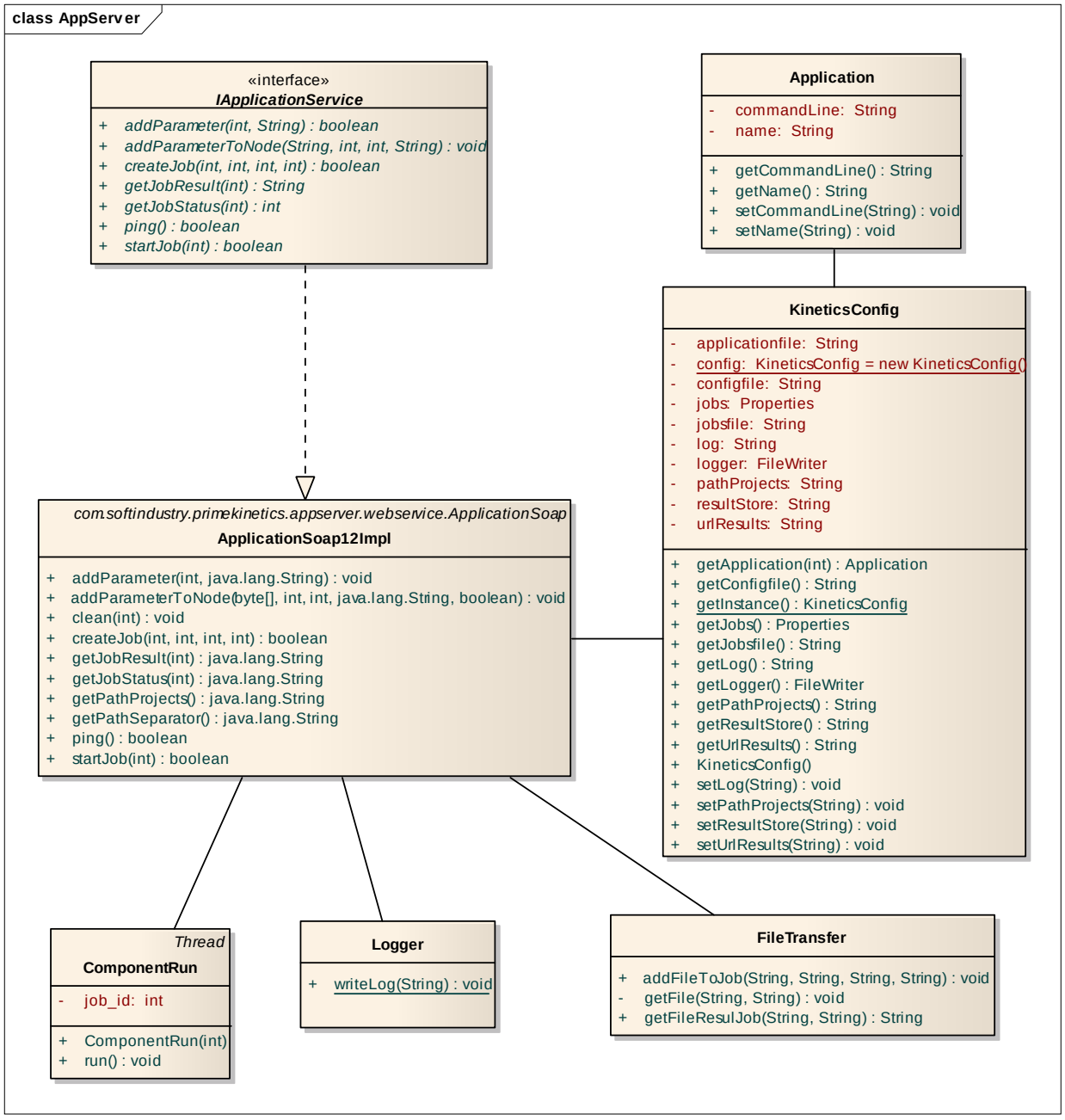

Figure 21. Application Server class diagram

IApplicationServer—the interface describes all the web-methods

Method	**Description**
addParameter	Receives the input information for specified node
addParameterToNode	Receives the files, which contain the work results of nodes connected with the specified ones
Clean	Clears the catalogue of node specified
createJob	Creates the new Job on server
getJobResult	Receives the work result of indicated job
getJobStatus	Receives the work result of specified job
getPathProjects	Returns the path to catalogue, in which all the files of current projects are stored
getPathSeparator	Returns the file separator for current OS(“/” for Unix or “\” for Windows)
startJob	Of indicated job starting for implementation

Application—contains the information about the scientific application

Method	**Description**
getCommnadLine	Returns the command line, which will start the applications
getName	Returns the name of scientific application
setCommandLine	Sets the command line
setName	Sets the application name

ComponentRun—is used for the scientific application starting. Each application is started in separate flow

Method	**Description**
Componentrun	The constructor, that receives as an input parameter job identifier
Run	Scientific application start

KineticsConfig—class, that provides the access to the main configuration files

Method	**Description**
getApplication	Returns the application by identifier
getConfigFile	Returns the path to the main configuration file
getJobsFile	Returns the path to the file, where the identifiers of current jobs are stored
getPathProjects	Returns the path to the catalogue where the information of current projects is stored
getUrlResult	Returns the url where the applications work results will be stored
getLogger	Returns the url on Logger class, which can be used for logs

5.8 Utility.dll structure

The Utilty.dll library in the server is used in executing remote applications. In this library the authorization, authentication, and data base work are managed. Its main structure is represented in Figure 22 and consists of the following parts:

Authentication module: Provides site's users authorization and authentication service on the application server.

1. Application Service—used to process scientific application requests.
2. JobService—used to process current jobs, and track execution status.
3. ComponentService—The service for component downloads.
4. WorkflowService—The service for work with the workflow.

5.8.1 Main modules

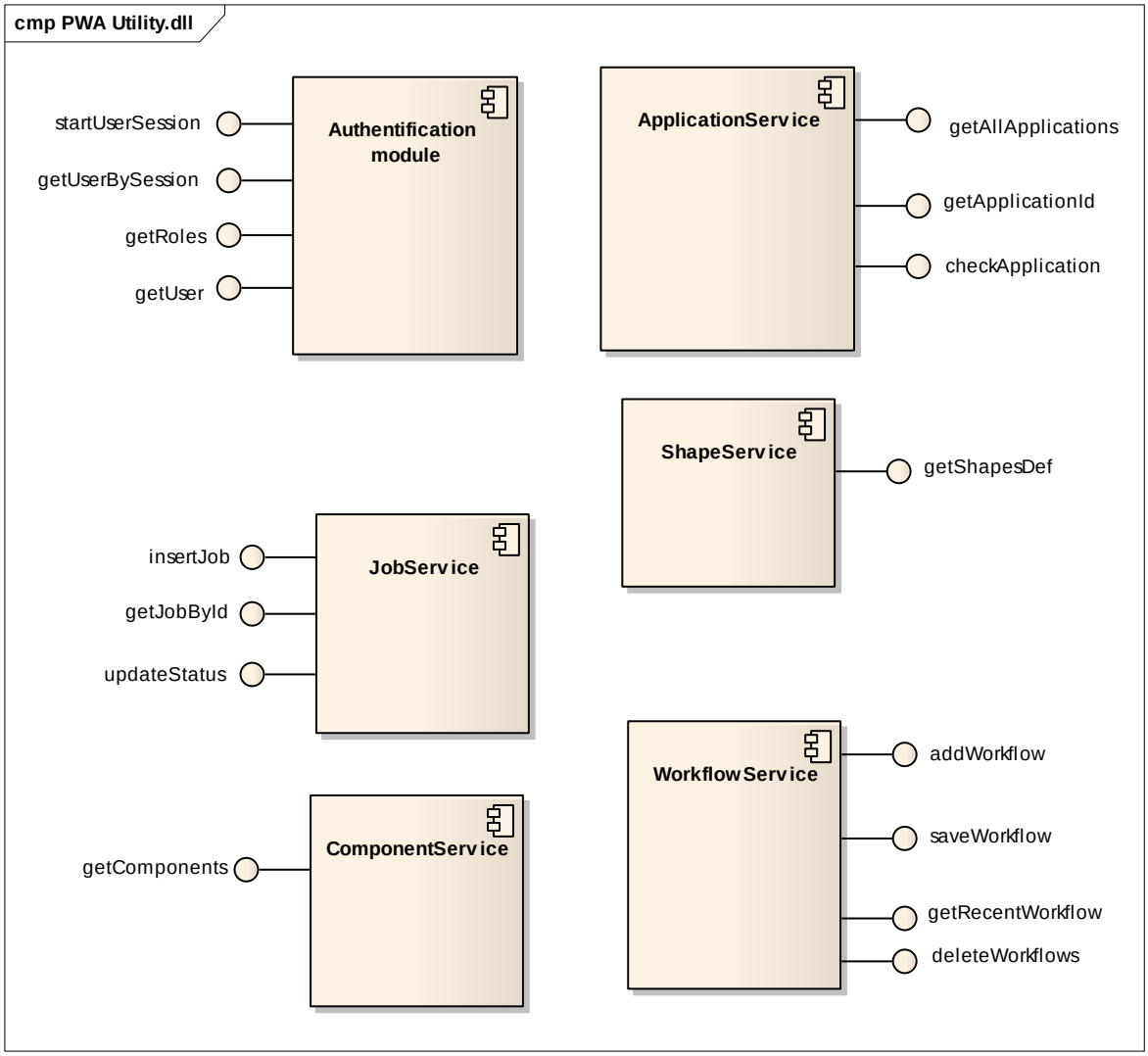

Figure 22. Utility.dll main modules

5.8.2 Classes diagram

The main library classes are presented in Figure 23.

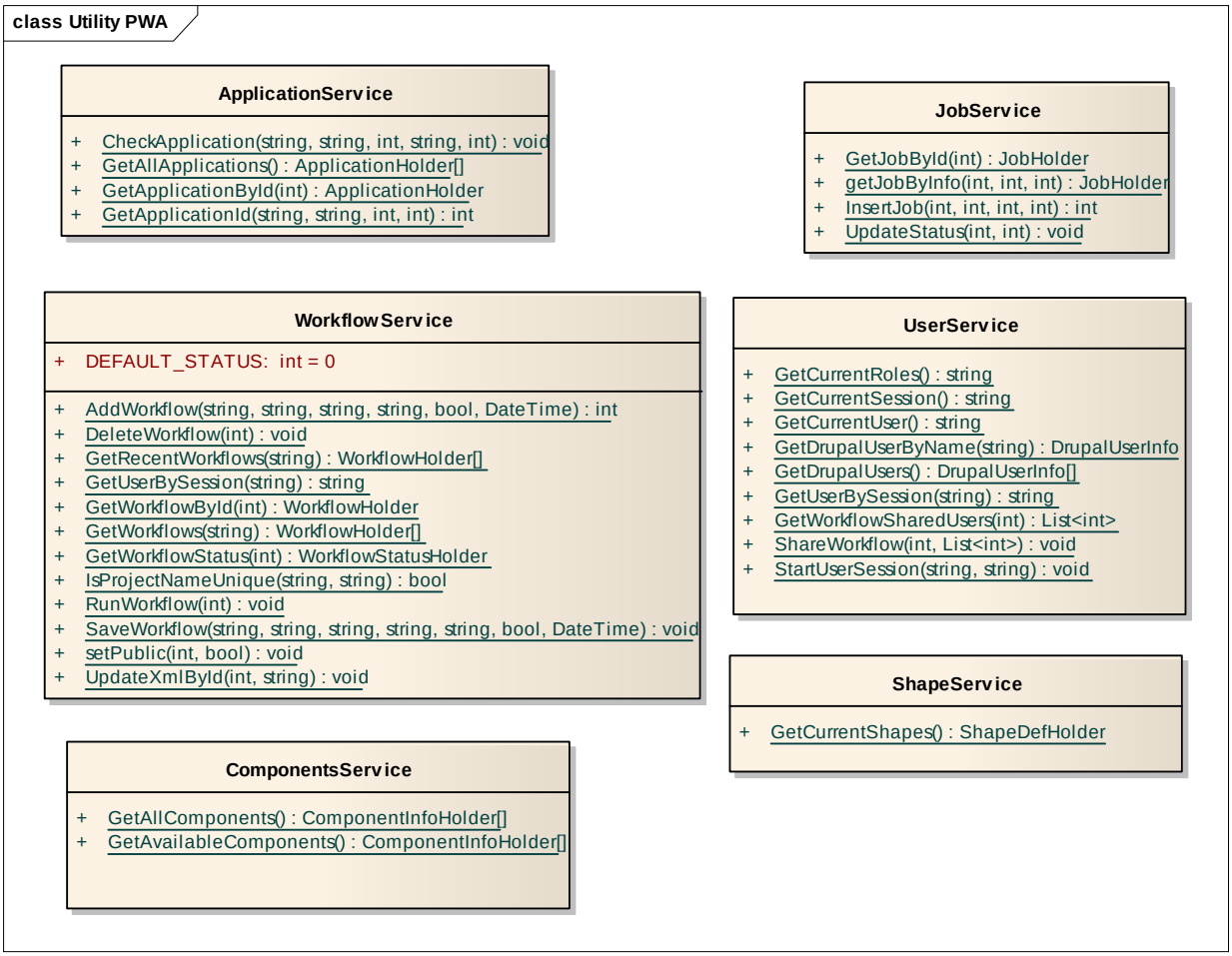

Figure 23. Utility.dll classes diagram

ApplicationService—the service for storing and receiving information, about the scientific applications

Method	Description
CheckApplication	Checks the whether the specified application exists in database or not
GetAllApplication	Receives all the available applications
GetApplicationById	Returns the applications by the identifier
GetApplicationById	Returns the application by name, host, and port on which it functions

JobService—the service for the work with the current tasks on applicationServer

Method	**Description**
GetJobById	Receives Job by the identifier
insertJob	Adds new Job
UpdateStatus	Job status update
getJobByInfo	Returns job by project identifier, node and application

UserService—in this class the service for work with user is implemented

Method	**Assignment**
GetCurrentRoles	Returns the current user roles
GetCurrentSession	Returns the session of current user
GetDrupalUsers	Returns all the users which are registered in the system
GetUserBySession	Returns the users on session
StartUserSession	Starts the new session for specified user

ShapeService—is used for the shapes download to PWA

Method	**Assignment**
GetCurrentShapes	Returns the shapes set as the xml-description

ComponentService—is used for clients component download

Method	**Assignment**
GetAllComponents	Returns the available components

WorkflowService—provides the work with diagrams

Method	**Assignment**
AddWorkflow	To add the new diagram
DeleteWorkflow	To delete the diagram
GetWorkflowById	To receive the diagram by the identifier
SaveWorkflow	To save the changes in diagram
setPublic	To make the diagram available for system users
getRecentWorkflows	To receive all the available diagrams

5.8.3 Database structure

The database structure is represented in Figure 24.

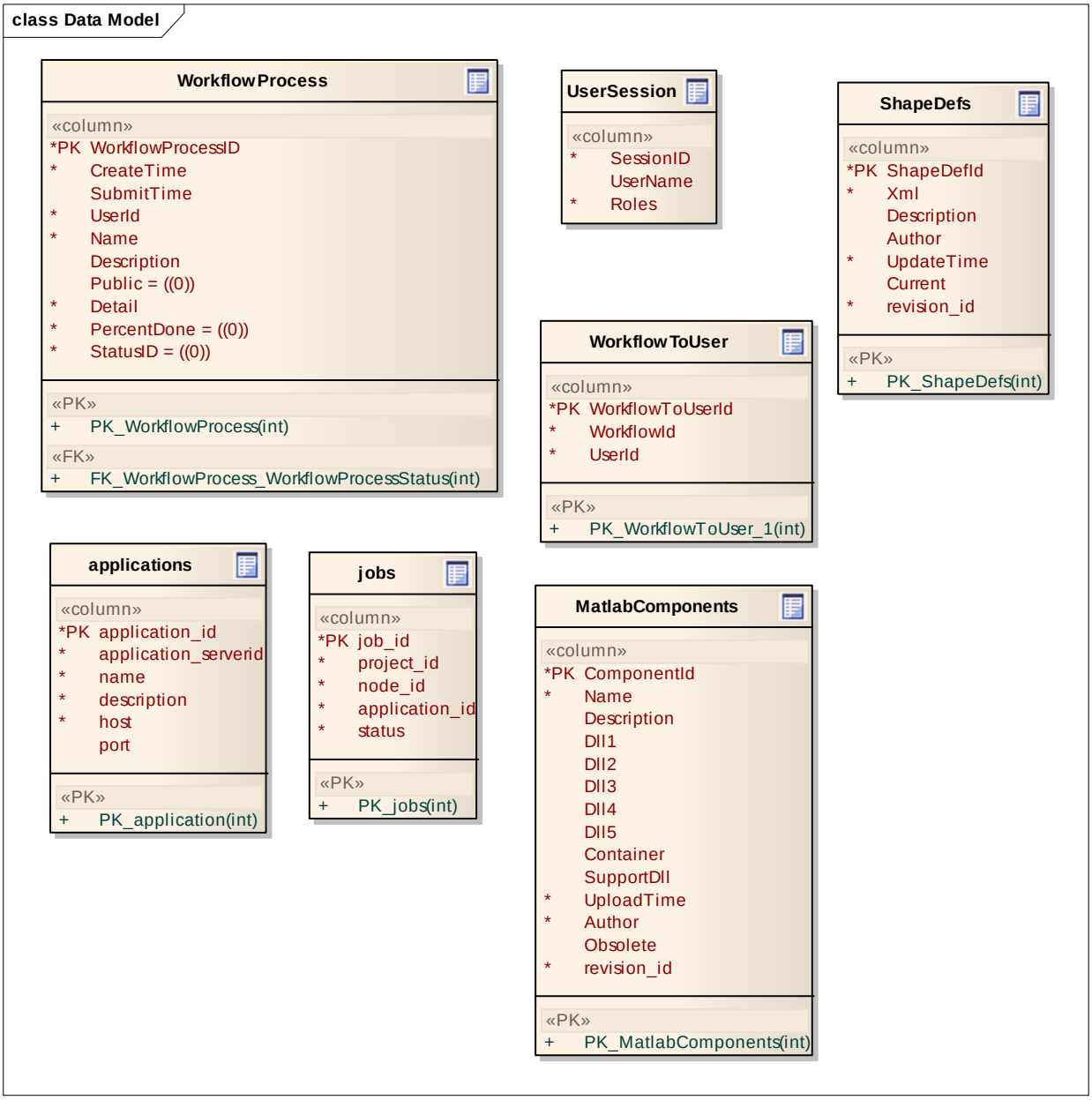

Figure 24. Database structure

ShapeDefs—shapes description

Method	**Description**
ShapeDefId	Shapes description Identifier
Xml	Xml-description shapes
Desciprion	Shapes description
Author	The author who created shapes
updateTime	The update time
Revision_id	The revision identifier, to which the shapes refer

MatLabComponents—component information

Method	**Description**
ComponentId	Component identifier
Name	Component name
Description	Component description
Dll1, Dll2, Dll3, Dll4, Dll5	Libraries names
SupportDll	Library name, that provides the interface of connection with PWA
Revision_id	Revision identifier, to which the component is referring
Author	The author who created the component
Obsolete	The indication of that the component is out of date
UploadTime	The component download time

WorkflowProcess—the information about the created projects

Method	**Description**
WorkflowProcessId	Unique project identifier
CreateTime	Creation date
SubmitTime	The last starting time
UserId	The user identifier, who created the project
Name	Project name
Description	Project description
Public	The indicator of that the project is available for all the users

UserSession—in this table the unique line that identifies the user's session is stored

Method	**Description**
SessionId	Session identifier
UserName	User's login
Roles	User's roles

Applications—the information about the servers' applications registered in the system

Method	**Description**
applicationId	Unique identifier of application server
Application_serverid	The identifier of applications on server
Name	Name
Description	Description
Host	Ip address or DNS host name
Port	The port on which the web-server is working

Jobs—the information about the current tasks on application servers

Method	**Description**
Job_id	Job unique identifier
Project_id	Project identifier
Node_id	The node identifier on diagram
Application_id	Applications identifier
Status	Implementation status

WorkflowToUser—the information about the user's access to the project

Method	**Description**
WorkflowToUserId	Unique identifier
WorkflowId	Unique project identifier
UserId	User identifier

5.9 Web-service Description

The PrIMeKineticsClient.dll library interacts with the server by means of web-service. The main methods used are represented below

Method	**Description**
GetWorkflow	Returns the workflow by identifier
DeleteWorkflow	Delete the workflow
GetAvailableComponents	To receive all the available components
GetCurrentShapes	To receive the current shapes
GetWorkflowSharedUsers	To receive the users, which have the access to the project
ShareWorkflow	To set the rights for the project access of specified users
SetPublicWorkflow	To make the project public
insertJob	To save new job on server
updateJobStatus	To update the job status
clearProduction	The deletion of all the components and resources
uploadProduction	The new recourses and components and shapes

6 Technologies used

6.1 PrIMe Portal

The PrIMe portal is executed using the PHP language with the help of CMF Drupal-5. The standard modules of Drupal core set are developed by third parties and obtained from the repository drupal.org. Part of the modules was modified specifically for the PrIMe portal.

The PrIMe portal uses MySQL for the database technology. It is working on web-server technology Apache2 under the OC Windows-2003 Server management.

6.2 Scientific Component Uploader and PrIMe Workflow Application

Both the SCU and the PWA utilize Microsoft .NET technologies. All code is written in the C# language. To enable the feature of native code implementation in the context of the client's browser Active X technology was used.

In the capacity of DBMS the MS SQL Server 2005 is used. It is run on web-server IIS under the management of OC Windows-2003 Server.

6.3 Application server

The application server utilizes Java technologies. For the creation of web-services the AXIS framework was used. It is run on tomcat 6 web-server.

7 Personnel Supported

This project supported mainly the programming consultant, Michael Gutkin, along with the Principal Investigator, Professor Michael Frenklach.

8 Publications and Presentations

1. “Cyber-enabled Data-centric Approach to Predictive Modeling,” M. Frenklach, NSF Workshop on Cyber-Enabled Discoveries and Innovations, (CDI) Initiative, Seattle, WA, November 11, 2007.
2. “PrIMe: Smart Science for Smart Combustion,” M. Frenklach, Z. M. Djurisic, A. Packard, D. M. Golden, W. H. Green, Jr., and P. J. Smith, 55th JANNAF Propulsion Meeting, Boston, MA, May 12-16, 2008.
3. “Analysis of Uncertainty in Model Prediction with Data Collaboration”, M. Frenklach, Massachusetts Institute of Technology, Department of Chemical Engineering, May 15, 2008.
4. “A New Platform for Predictive Modeling: PrIMe”, M. Frenklach, John Zink Company, June 5, 2008.
5. “PrIMe: Workflow Architecture,” M. Frenklach, M. Gutkin, and P. J. Smith, Work-in-Progress Poster W5P040, 32nd International Symposium on Combustion, Montreal, Canada, August 3-8, 2008.
6. “A New Platform for Predictive Modeling: PrIMe,” M. Frenklach, Z. Djurisic, and M. Gutkin, Multiagency Coordination Committee for Combustion Research (MACCCR)—Fuels Research Review, Gaithersburg, MD, September 8-10, 2008.
7. “PrIMe: Why, What, How,” M. Frenklach, NRC Committee on Cyberinfrastructure for Combustion, The National Academies, Washington, D.C., March 9-10, 2009.
8. “Integration of DataCollaboration with PrIMe,” X. You, T. Russi, D. Yeates, A. Packard, and M. Frenklach, 6th U.S. National Combustion Meeting, Ann Arbor, MI, May 17-20, 2009, Paper No. 12G4.

9 Significant Interactions

In collaboration with Dr. William Roquemore (USAF AFMC AFRL/RZ), Dr. Meredith Colket (UTRC), and Professor Hai Wang (USC) we establish a PrIMe Work Group for the SERDP project. The purpose of the group is to exchange the information and data on the ongoing collaborative research on soot formation. The group is currently administered by Dr. Colket.